Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2009

Carlos Miguel Ferreira

Co-Processador de Hardware para o Executivo de Tempo-Real OReK

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2009

Carlos Miguel Ferreira

Co-Processador de Hardware para o Executivo de Tempo-Real OReK

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Arnaldo Silva Rodrigues de Oliveira, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president	**Doutor Valeri Skliarov** Professor Catedrático da Universidade de Aveiro
vogais / examiners committee	**Doutor Arnaldo Silva Rodrigues de Oliveira** Professor Auxiliar da Universidade de Aveiro (orientador)
	Doutor António José Duarte Araújo Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto (arguente)

agradecimentos /
acknowledgements

Desde já, começo por agradecer ao meu orientador Professor Doutor Arnaldo Silva Rodrigues de Oliveira, que me ajudou e orientou e que cujas críticas construtivas ajudaram a aprumar e melhorar os conteúdos desta dissertação. Agradeço também aos meus pais, que tanto esforço, dedicação e sacrifico depositaram em mim, para que eu hoje pudesse chegar onde estou, e agradeço ainda ao Mestre Nelson Silva pelas dicas e ajudas oferecidas, que muito precisas se tornaram! E por fim, agradeço-te Senhor, por toda a calma que me deste, por todas as dificuldades que me ajudas-te a superar e por toda esta capacidade de raciocínio que construíste dentro de mim, pois sem ti, nada disto teria sido possível.

A todos os outros que me possa ter esquecido, a vocês agradeço toda a cooperação e apoio! Obrigado!

Qualquer dúvida, crítica constructiva ou observação, poderá ser remetida para: carlosmf.pt@gmail.com

Resumo

Esta dissertação, apresenta a implementação de um co-processador de utilização genérica para o executivo de tempo-real OReK.
Ao longo dos 5 capítulos que constituem esta dissertação, são apresentados os objectivos do trabalho, são relembrados alguns conteúdos teóricos que fazem parte do desenvolvimento do projecto, são apresentados todos os constituintes do co-processador explicando o seu funcionamento, sendo por fim demonstrado através da avaliação temporal, como a utilização de um co-processador para aceleração das funcionalidades de um executivo de tempo-real, poderá efectivamente melhorar o seu desempenho e determinismo.
Os objectivos principais deste trabalho passam por, estudar as ferramentas e plataformas necessárias ao seu desenvolvimento, especificar e projectar a arquitectura do co-processador assim como todas as suas funcionalidades internas, adaptação do executivo OReK para albergar o funcionamento em conjunção com o co-processador desenvolvido, sendo por fim, feita a validação e avaliação do sistema.
Através dos resultados da avaliação do sistema com a utilização do co-processador, é demonstrado que existe um forte melhoramento do determinismo e também do desempenho do executivo OReK. No entanto, é também demonstrado como estes resultados podem ser postos em causa, caso exista uma elevada latência, na interligação entre o processador e o co-processador, o que pode atrasar a execução de todo o sistema, podendo até mesmo prejudica-lo. Todos estes aspectos são estudados, desenvolvidos e explicados ao longo desta dissertação.

Abstract

This Master Thesis, presents the implementation of a generic use coprocessor for the real-time kernel OReK.
For the next 5 chapters, which constitute this thesis, the work objectives are presented, the necessary theoretical contents for the project development are remembered, the coprocessor internal constitution is presented and explained, and in the end, it is demonstrated through temporal evaluation that, the use of a coprocessor can in fact, accelerate the real-time kernel functions, improving it's performance and determinism.
This work main objectives are, to study the tools and platforms necessary for the project development, specify and develop the coprocessor architecture with all the necessary internal functions, OReK kernel adaptation in order to use the coprocessor, and in the end, to validate and test the system.
The system evaluation results, using the coprocessor, demonstrated the existence of a serious improvement in the determinism and performance of the OReK kernel. Meanwhile, it is also demonstrated how these results can be rendered useless, if there is a high latency in the connection between the main processor and the coprocessor, which can delay the system execution and even harming it. All of these aspects are studied, developed and explain in the course of this master thesis.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Neste capítulo, irá ser feita a apresentação dos conteúdos desta dissertação. Será feita uma breve descrição dos actuais sistemas embutidos e sistemas integrados reconfiguráveis. Depois, será explicado quais os motivos que levaram à concretização desta dissertação, seguindo de como será feita a demonstração e prova do conceito, acompanhando dos objectivos a atingir. No final, será apresentado um resumo da estrutura da dissertação.

1.1 Sistemas Embutidos

Com o avanço da tecnologia, temos observado um aumento exponencial da utilização de sistemas embutidos, os quais são sistemas computacionais dedicados a uma dada aplicação. Estes sistemas embutidos, são utilizados de forma diária em massa pelo público geral em telemóveis, leitores de música portáteis e equipamentos domésticos. São no entanto, usados também em sistemas críticos como meios de transporte públicos, equipamentos hospitalares e ainda sistemas de telecomunicações.

Sistemas embutidos são normalmente desenhados e construídos para um determinado fim específico, sendo que por esta razão, constituídos por um conjunto estritamente necessário de funcionalidades e recursos. Estes podem ser constituídos na sua versão mais simples por um micro-processador, memória *RAM* e memória *Flash*, ou então nas suas versões mais complexas, podendo ter vários micro-processadores, controladores, vários tipos de memória e periféricos de comunicação, conseguindo interagir com outros sistemas.

Tipicamente, um sistema embutido combina elementos de hardware com aplicações de software, aproveitando as vantagens de ambos. Assim sendo, estes sistemas contêm um número mínimo de recursos de hardware, necessários apenas para o seu funcionamento, limitando no entanto as suas capacidades de expansão. Este facto é tido em conta ao desenvolver as aplicações, sendo estas optimizadas o melhor possível, pois normalmente, os sistemas embutidos têm uma quantidade muito limitada não só de memória como também de processamento, existindo também a preocupação do consumo energético.

1.1.1 Sistemas Integrados

Sistemas integrados, também conhecidos pelo seu termo inglês System on Chip (SoC), surgiram como resultado dos sucessivos avanços tecnológicos da micro electrónica dando

como exemplo, a redução do tamanho de um transístor. Estes avanços tecnológicos, permitem actualmente a implementação num único substracto de silício, de todos os componentes necessários para construir um sistema computacional complexo. A possibilidade de poder incorporar, vários componentes como processadores, co-processadores, memória, controladores de comunicações ou outros periféricos, permite tirar partido da sua proximidade, desde reduzir a dimensão do circuito como aumentar a velocidade do seu funcionamento, reduzir consumo de potência e ainda permitir a redução do custo de produção unitário.

No entanto, o aumento da complexidade de um sistema integrado, traz desvantagens e coloca grandes desafios na sua fase de desenvolvimento, teste e validação, quer devido ao facto da produtividade do projecto nem sempre acompanhar o aumento da sua complexidade, quer pela necessidade de reduzir o tempo de desenvolvimento e produção de novos produtos e sistemas.

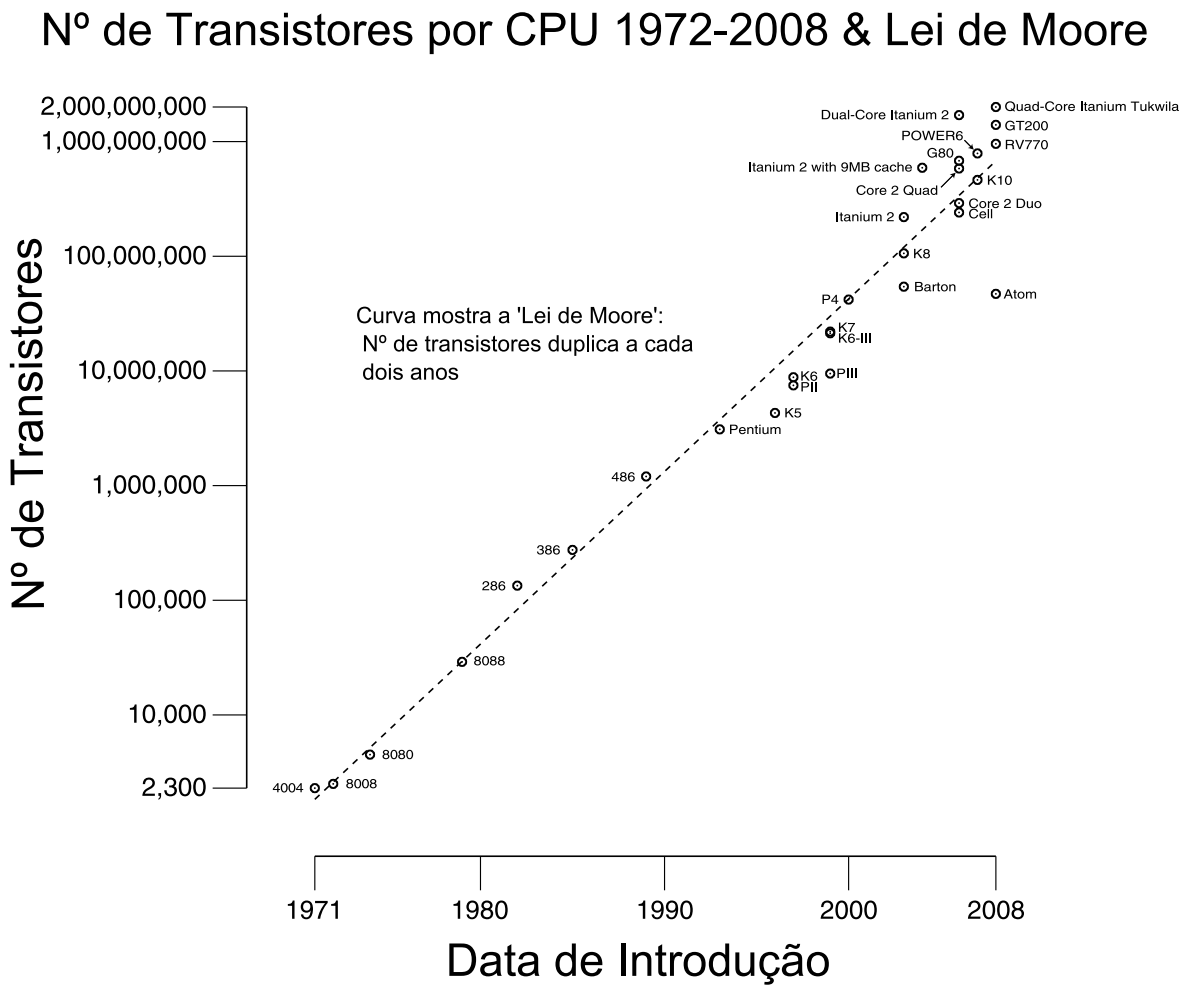


Figura 1.1: Comparação da Lei de Moore com o Número de Transístores dos Processadores Actuais.

Além disso, tradicionalmente os SoCs não podem ser modificados do ponto de vista físico, uma vez fabricados, não podendo assim corrigir, quaisquer erros que possam existir, sendo necessário o projecto e fabrico de novos circuitos com todos os custos associados,

sendo que na grande maioria devido a estarem soldados a placas, não podem ser trocados de uma forma simples, tornando a sua substituição complexa e de elevados custos.

Para reduzir o tempo de desenvolvimento, recorre-se cada vez mais a linguagens abstractas de desenvolvimento, numa tentativa de criar modelos iniciais do sistema, sendo também, reutilizados módulos já pré-desenhados e validados.

Por estas razões, o desenvolvimento de circuitos integrados para aplicações específicas, só se torna rentável para mercados de grande dimensão, impossibilitando os desenvolvimentos para pequenos nichos de mercados.

Na secção seguinte irá ser explicado como se conseguiu responder à necessidade de desenvolver circuitos integrados para aplicações específicas a baixo custo.

1.1.2 Sistemas Reconfiguráveis

Com a evolução dos circuitos integrados, passou a ser possível construir circuitos lógicos programáveis pelo utilizador, denominados de FPLD *(Field Programmable Logic Devices)*.

Dos dispositivos programáveis existentes, os mais poderosos e relevantes para o enquadramento desta dissertação, são as FPGA's *(Field Programmable Gate Arrays)*. As FPGA's, não são mais do que matrizes de blocos lógicos programáveis, interligados entre si por ligações que também são programáveis. Cada bloco lógico programável, pode ser constituído por vários tipos de elementos tais como, tabelas de verdade, *multiplexers*, portas lógicas, flip-flops, memória, entre outros. Na periferia da FPGA, existem blocos de entrada/saída programáveis que permitem fazer a interface entre os blocos lógicos e os pinos de saída. Os recursos de

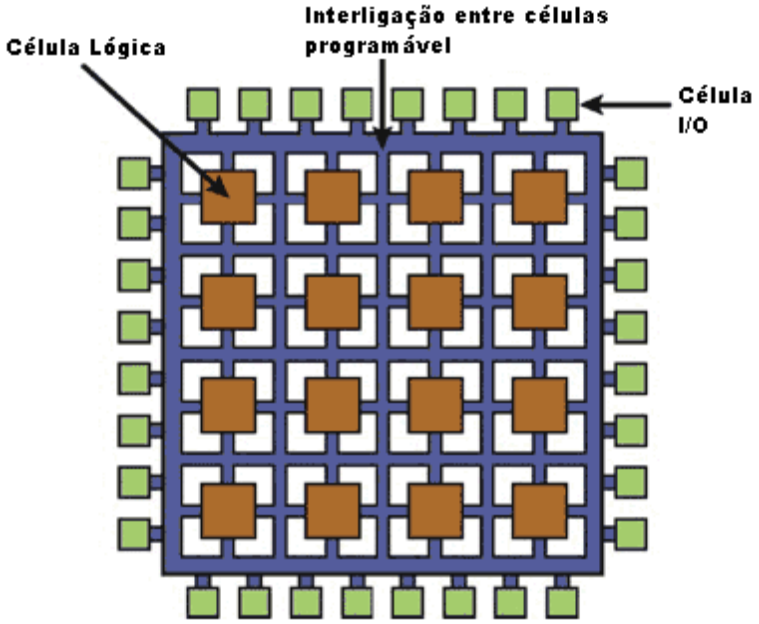


Figura 1.2: Arquitectura interna de uma FPGA

uma FPGA, são normalmente controlados pelos bits de uma memória de configuração, que ao ser reprogramável, é possível alterar o circuito lógico implementado, ou seja, o sistema poderá ser reconfigurado. Sendo assim, as FPGA's tornam-se as plataformas de eleição para

a implementação de sistemas reconfiguráveis e assim, torna-se possível implementar processadores dedicados para uma dada aplicação, explorando o paralelismo e aumentado a velocidade de processamento.

Ao permitir a criação de processadores ou circuitos para executar tarefas específicas, as FPGA's permitem desenvolver sistemas integrados para mercados de pequenas dimensões, o que antes não era possível devido aos elevados custos, como já foi descrito anteriormente. Sendo assim, devido ao elevado número de unidades produzidas, os custos de produção destes dispositivos reconfiguráveis tornam-se baixos e acessíveis.

Algo que deve ser referido, é o facto das FPGA's permitirem que erros existentes em projectos de sistemas que nelas se baseiem, possam ser solucionados mais tarde e consequentemente, ser possível carregar um novo ficheiro de configuração para a memória da FPGA, reconfigurando o circuito para uma versão actualizada. Os custos destas actualizações, são apenas os de desenvolvimento do projecto, pois não é necessário alterar ou modificar qualquer elemento físico.

Em suma, seguindo as razões acima referidas, verificamos que o desenvolvimento e implementação de projectos para áreas específicas, pode ser executado utilizando FPGA's, garantindo assim um baixo custo de execução e também de manutenção.

1.2 Enquadramento e Motivação

Esta dissertação está enquadrada em duas áreas de investigação e desenvolvimento, a área de circuitos reconfiguráveis e a área de sistemas de tempo-real críticos. Este trabalho, é de todo o interesse, de quem trabalhar com sistemas de tempo-real, implementados em circuitos reconfiguráveis, e que requer um elevado nível de determinismo, obtendo ao mesmo tempo um elevado nível de desempenho.

Uma quantidade importante dos sistemas embutidos, possui características de tempo-real. Sistemas embutidos como controladores de temperatura, de *airbags*, de elementos atmosféricos, de sistemas mecânicos complexos ou até mesmo outros sistemas de utilização mais geral como telemóveis, leitores de musica portáteis e até mesmo elevadores, todos estes sistemas têm requisitos e características bastante específicas, dando como exemplo, a necessidade de verificar periodicamente a temperatura interna de um reactor, analisar e consequentemente, tomar uma decisão. Sendo assim, para manter o seu bom funcionamento e por forma a que desempenhem bem as suas funções, estes sistemas por norma utilizam um executivo de tempo-real, para gerir a execução das tarefas de acordo com parâmetros temporais ou de activação muito restritos.

Os executivos de tempo-real, ao contrário de outros sistemas de uso genéricos, presam-se por ser determinísticos e também pela sua elevada precisão temporal. São sistemas que optam por executar as suas tarefas com o intuito de apenas obter um resultado, no instante de tempo em que realmente é necessário, contrariando assim os executivos genéricos, que optam por obter um resultado, no menor curto espaço de tempo. Sendo assim, devido ao facto de um sistema de tempo-real ser determinístico e ter o forte controlo sobre a temporização da execução das tarefas, torna-se mais adequado para sistemas críticos quando comparado com a utilização de um sistema genérico.

Na área de sistemas de tempo-real, podemos encontrar diversos executivos, estando a sua programação e aperfeiçoamento directamente relacionados com as tarefas a desempenhar. No entanto, todos os executivos implementados em software sofrem de um problema comum,

a falta de determinismo.

Este indeterminismo, é provocado muitas das vezes, pelas constantes paragens de processamento das tarefas, para execução das funcionalidades do executivo. Dando como exemplo, existirem muitas interrupções provenientes de fontes não determinísticas, o executivo irá fazer várias paragens no processamento das tarefas, para avaliar as suas opções e tomar decisões. Estas paragens, podem provocar falhas nos sistemas mais críticos, podendo levar a catástrofes.

É nesta base, que se fundou a motivação para a investigação e desenvolvimento do projecto, que culminou com a elaboração desta dissertação. Tendo como por base, o executivo OReK(*Object-Oriented Real-Time Kernel*), procurou-se desenvolver um co-processador numa plataforma contendo um circuito reconfigurável, que interligado à componente em software do executivo, implementar em hardware, as suas diversas funcionalidades, de forma a que permitisse aumentar de forma significativa o determinismo, aumentando também o desempenho do processamento das tarefas.

1.3 Objectivos

Para a o desenvolvimento da componente de projecto e elaboração desta dissertação, estabeleceu-se a seguinte lista de objectivos:

1. Estudo de conceitos sobre sistemas de tempo-real e familiarização com a interface de programação típica dos executivos de tempo-real tendo como exemplo o OReK.

2. Estudo das funcionalidades e interface do processador PowerPC 405 embutido nas FPGA's da Xilinx, para o qual será desenvolvido o co-processador de suporte ao OReK.

3. Estudo da plataforma de desenvolvimento composta por uma placa com uma FPGA de elevada capacidade da Xilinx, contendo o processador PowerPC 405, e pelos respectivos ambientes integrados de projecto de hardware e de software.

4. Especificação das funcionalidades, arquitectura e parâmetros do co-processador, desenvolvimento do mesmo usando os ambientes integrados de síntese de hardware a partir de modelos elaborados com a linguagem VHDL e sua integração num SoC baseado em FPGA.

5. Adaptação do executivo de tempo-real OReK de forma a tirar partido e abstrair as funcionalidades disponibilizadas pelo co-processador desenvolvido.

6. Teste e avaliação do sistema.

Método de Demonstração

Para se demonstrar como a utilização de um co-processador, pode efectivamente aumentar melhorar de forma significativa o funcionamento de um executivo de tempo-real, é necessário proceder à sua implementação, de forma a que se possa testar e comparar com o executivo sem aceleração por hardware.

Sendo assim, decidiu-se utilizar uma FPGA Virtex 2 Pro da empresa Xilinx, como plataforma de desenvolvimento do co-processador, utilizando a linguagem VHDL para proceder

à sua descrição e estruturação. O software de desenvolvimento utilizado, será o ISE e o EDK, ambos pertencentes também à empresa Xilinx.

Desta forma, irá ser possível analisar e avaliar o desempenho do co-processador de forma detalhada e exacta, e assim, comprovar que a utilização de um co-processador para aceleração das funcionalidades de um executivo de tempo-real, efectivamente aumenta o seu determinismo significativamente, como também permite aproveitar de uma forma mais eficiente os recursos do processador.

1.4 Organização da Dissertação

Esta dissertação está organizada em:

- **Capítulo 2** - Conceitos Teóricos e Pesquisa Relacionada
  - Apresentação de conteúdos teóricos relevantes e dentro do âmbito da dissertação.
  - Apresentação de trabalhos executados na área de outros autores, nos últimos cinco anos.
  - Tabela de comparação de co-processadores ou plataformas de desenvolvimento de co-processadores, para sistemas de tempo-real.
- **Capítulo 3** - Implementação do Co-Processador
  - Apresentação do co-processador para o executivo de tempo-real OReK.
  - Especificação dos requisitos necessários à implementação da arquitectura do co-processador.
  - Explicação em detalhe, da arquitectura de cada elemento interno do co-processador.
- **Capítulo 4** - Avaliação e Desempenho
  - Apresentação da plataforma de teste e configuração usada.
  - Explicação de como foi feita a avaliação temporal do executivo.
  - Avaliação temporal e apresentação dos resultados obtidos.
- **Capítulo 5** - Conclusão
  - Resumo da Implementação.
  - Conclusão final dos resultados obtidos.
  - Possíveis trabalhos futuros.
- **Apêndice A** - O Executivo de Tempo-real OReK
  - Descrição detalhada do executivo de tempo-real OReK na sua implementação em *software*.

Capítulo 2

Conceitos Fundamentais e Trabalho Relacionado

Neste capítulo irá ser feita a apresentação de alguns conceitos fundamentais sobre sistemas de tempo-real. São apresentados e explicados, alguns problemas que surgem quando são utilizados sistemas genéricos vulgares em determinadas actividades, e que estes não conseguem resolver de uma forma adequada, existindo a necessidade de projectar e desenvolver um sistema de tempo-real adequado ao caso.

É descrita a constituição de um sistema de tempo-real, explicando a necessidade de ter um executivo de tempo real que mantém um rigoroso controlo sobre a temporização e a ordem em que são executadas as suas actividades, mostrando as consequências que poderão existir, em caso de falha.

Por forma a completar este capítulo, são apresentados trabalhos relacionados com o tema desta dissertação. Estes trabalhos abordam aspectos que se interligam com o tema desta dissertação, como a redução do tempo necessário para o processamento das rotinas internas do executivo (*overhead* do executivo), a implementação de escalonadores de tarefas em *hardware*, para redução do tempo que demora a obter a tarefa, a implementação de um controlador que permite fazer a detecção de *deadlocks*[CES71] e por fim, uma avaliação do desempenho entre o executivo MicroC/OS-II e uma unidade de tempo-real.

São também apresentados outros trabalhos, não directamente relacionados com o tema desta dissertação mas que no entanto abordam a questão do suporte por *hardware*, da execução de aplicações ou tarefas. É também descrito um implementação em hardware, de um analisador de código que permite às aplicações, fazer uma análise da sua própria execução.

Por fim, será apresentada uma tabela de comparação de vários co-processadores para executivos de tempo-real já implementados ou analisados, identificando as suas capacidades, qualidades sendo também feita, uma pequena descrição.

2.1 Sistemas de Tempo-Real

A existência de sistemas de tempo-real, provém da necessidade de dar resposta a problemas, aos quais os sistemas normais não têm capacidade para resolver. Um sistema de uso genérico, ao contrário de um sistema de tempo-real, não tem a capacidade de controlar de uma forma precisa, a temporização da execução das suas tarefas, sendo geralmente projectados e construídos de uma forma mais complexa, tendo como principal objectivo a execução das suas tarefas, no menor curto espaço de tempo. Um sistema de tempo-real é normalmente de construção extremamente simples, não tendo que necessariamente ser rápido, devendo sim, ser o mais determinístico e preciso na sua execução.

Sistemas de tempo-real (figura 2.1), podem ser encontrados em diversas utilizações, como

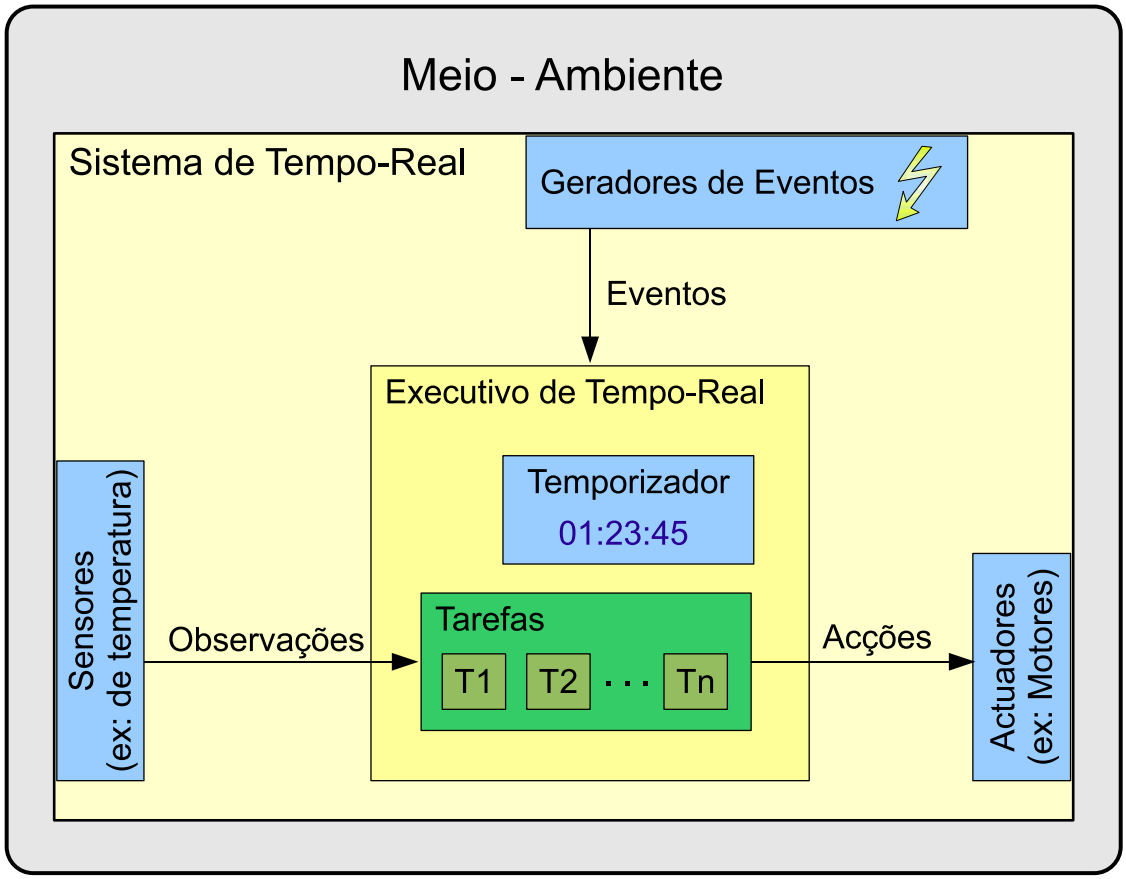


Figura 2.1: Exemplo de um Sistema de Tempo-Real no Meio-Ambiente.

controladores de temperatura em caldeiras de aquecimento de água, alarmes de segurança, equipamentos de telecomunicações (telemóveis, *switches, routers*) ou ainda equipamentos multimédia (mesas de mistura, dispositivos de música portáteis).

Os sistemas referidos, são apenas alguns exemplos de sistemas onde é fulcral, a necessidade de executar as suas tarefas de uma forma continua e acompanhada, estando presente a questão da precisão temporal. A precisão temporal garante que uma tarefa cumpre as suas restrições temporais, sendo estes, os indicadores temporais que definem o momento de activação, periodicidade de execução e também o momento de terminação.

Como já foi indicado, para além da precisão temporal existe também a preocupação do determinismo do sistema. Ser determinístico, significa que as suas acções poderão ser previsíveis, o que no caso do sistema de tempo-real, isto significa que a sua actividade pode ser facilmente prevista ou mesmo simulada para questões de testes. Isto garante a própria fiabilidade do sistema, mesmo antes de ele ser posto em execução, o que em muitos casos poderá ser complexo ou financeiramente custoso, dando como exemplo, o lançamento de foguetões espaciais.

Uma má execução das tarefas de um sistema de tempo real, poderá degradar no mínimo os resultados obtidos, podendo até um caso mais crítico, provocar danos consideráveis ao meio-ambiente onde estes sistemas estão inseridos. Utilizando como exemplo, os sistemas acima referidos, caso existisse uma falha na execução das tarefas, a caldeira da água poderia ser destruída devido ao excesso de pressão, um alarme de segurança poderia não detectar uma entrada ilícita, *switches* e *routers* poderiam perder pacotes de informação levando à degradação ou perda completa do acesso à rede, ou até mesmo, uma degradação na qualidade sonora, como é o caso de uma mesa de mistura ou do leitor de musica portátil.

Sendo assim, um sistema de tempo-real pode ser classificado de acordo com as suas restrições temporais:

- **Não crítico (*Soft Real-Time*)** - O sistema apresenta restrições temporais do tipo Firme (*Firm*) ou Suave (*Soft*).
- **Crítico (*Hard Real-Time*)** - O sistema apresenta pelo menos uma restrição do tipo Rígida (*Hard*).

2.1.1 Modelo de Programação Baseado em Tarefas

Em sistemas operativos, o termo tarefa (ou processo) é utilizado para descrever um conjunto de instruções que representa a sua actividade, um contexto de execução formado pelos registos do processador, parâmetros e atributos que variam de sistema para sistema, residindo num espaço de memória próprio que poderá ou não ser partilhado.

Por forma a interagir com o meio-ambiente e tomar acções, os sistemas de tempo-real são normalmente sistemas multi-tarefa, sendo estas tarefas controladas e geridas pelo executivo. Consequentemente, cada tarefa pode encontrar-se em vários estados de execução, estando estes representados na figura 2.2, estando abaixo descritos cada um de um forma mais detalhada.

- **Pronta a executar (*Ready*)** - Tarefa está pronta para executar, aguardando que o executivo a coloque em execução, no contexto do processador.
- **Em execução (*Running*)** - Tarefa encontra-se em execução.
- **Bloqueada (*Blocked*)** - Tarefa encontra-se bloqueada e aguarda a libertação de um recurso ou sinalização de execução.
- **Suspensa (*Suspended*)** - Tarefa encontra-se suspensa e aguarda que seja resumida.
- **Dormente (*Idle*)** - Tarefa encontra-se dormente e aguarda que seja activada.

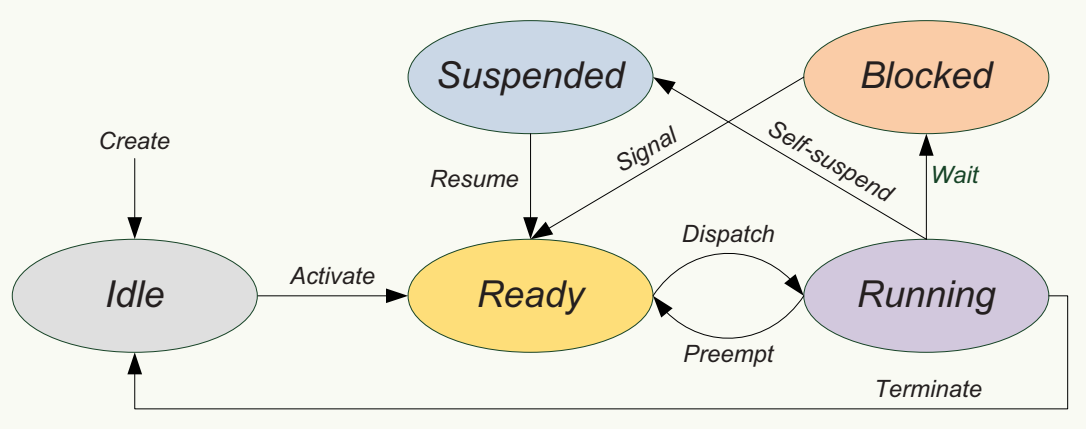


Figura 2.2: Estados possíveis de uma tarefa.

2.1.1.1 Parâmetros de uma Tarefa

As tarefas, por forma a que a sua execução ao longo do tempo de actividade do sistema, seja gerida de uma forma precisa, contêm parâmetros temporais que determinam que estabelecem quando devem deverão iniciar a sua actividade, quando é que deverão ser colocadas em execução e ainda, quando é que o resultado das suas acções deverá estar pronto, ou seja, quando é que deverão terminar.

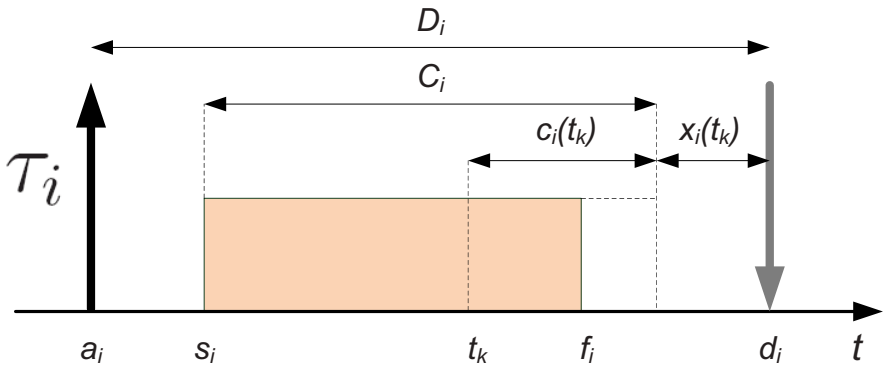


Figura 2.3: Parâmetros temporais de uma tarefa.

Uma tarefa τ_i é caracterizada por diversos parâmetros:

- a_i - Instante de Activação.
- s_i - Instante de Execução.
- f_i - Instante de Terminação.
- d_i - *Deadline* Absoluto.
- D_i - *Deadline* Relativo.
- C_i - Máximo tempo de Execução no Pior Caso.
- $c_i(t_k)$ - Tempo Máximo de Execução Residual no instante t_k.
- $x_i(t_k)$ - Tempo Mínimo de Relaxamento (folga) no instante t_k.

– Podemos calcular para um dado instante t_k, o tempo mínimo de relaxamento, utilizando a formula

$$x_i(t_k) = d_i - [t_k + c_i(t_k)]$$

sendo que para $x_i(t_k) < 0$ significa que a tarefa poderá não respeitar o seu *deadline* absoluto (d_i).

2.1.1.2 Criticalidade

A criticalidade das tarefas, poderá ser caracterizada da seguinte forma:

- **Rígidas (*Hard Real-time*)** - O não cumprimento das restrições temporais, poderá resultar num desastre catastrófico para o sistema e o meio envolvente.
- **Firmes (*Firm Real-time*)** - O não cumprimento das restrições temporais, poderá resultar numa perda total da utilidade do resultado obtido pela tarefa.
- **Suaves (*Soft Real-time*)** - O não cumprimento das restrições temporais, poderá resultar numa degradação do resultado obtido pela tarefa.
- **Ordinárias (*Non Real-time*)** - Não possui restrições temporais, não sendo o seu desempenho critico para o sistema.

2.1.1.3 Periodicidade e Modo de Activação

Quanto à métrica temporal, as tarefas podem ser caracterizadas nos seguintes tipos:

- **Periódicas** - As tarefas são activadas segundo um período de tempo definido.
- **Esporádicas** - Tarefas que são activadas de forma assíncrona mas que no entanto respeitam um intervalo de tempo mínimo entre activações.
- **Aperiódicas** - Tarefas que poderão ser activadas em qualquer momento, não respeitando um intervalo mínimo entre activações. Tarefas aperiódicas também podem ser vistas como uma forma de tarefas esporádicas, mas com um intervalo mínimo de activação igual a zero.

2.1.1.4 Requisitos

As restrições das tarefas podem ser de vários tipos:

- **Temporais** - Definem parâmetros temporais como, instante inicial de activação, período, deadline e intervalo mínimo entre activações.
- **Precedência** - Estabelecem uma ordem de execução das tarefas.
- **Utilização de Recursos** - utilização de recursos de forma concorrente por parte das tarefas (será discutido ainda neste capítulo).

2.1.1.5 Preempção

Uma tarefa admite preempção quando a sua execução, puder ser interrompida para que se possa executar uma outra tarefa de prioridade mais elevada. Caso um conjunto de tarefas admita preempção em qualquer altura da sua execução, diz-se que o conjunto de tarefas admite preempção total.

2.1.2 Escalonamento de Tarefas

A ordenação de tarefas de acordo com uma prioridade pré-definida, denomina-se por escalonamento de tarefas. Isto permite criar um alinhamento para execução das tarefas pelo processador, de forma a satisfazer as restrições temporais e produzir assim, um escalonamento praticável. Sendo assim, um conjunto de tarefas diz-se escalonável caso exista pelo menos um escalonamento praticável.

2.1.2.1 Políticas de Escalonamento

Para que um sistema de tempo-real, possa garantir o cumprimento das restrições temporais, necessita de utilizar políticas de escalonamento adequadas. De seguida, irão ser apresentadas apenas, as políticas de escalonamento que se enquadram no âmbito desta dissertação.

Escalonamento Baseado em Prioridades

O escalonamento baseado em prioridades, utiliza as restrições temporais das tarefas para calcular a prioridade de cada uma. Assim, a execução das tarefas é feita de acordo com a prioridade de cada uma.

Estes são os algoritmos mais utilizados:

- **Algoritmo *Rate Monotonic* -** Este algoritmo atribui uma prioridade fixa inversamente proporcional ao seu período. Quanto menor for o período, maior é a sua prioridade e consequentemente, menor é o seu atraso de execução.
- **Algoritmo *Deadline Monotonic*** - Este algoritmo atribui uma prioridade fixa inversamente proporcional ao seu *deadline* relativo. Quanto menor for o *deadline* relativo, maior é a sua prioridade e consequentemente, menor é o seu atraso de execução.
- **Algoritmo *Earliest Deadline First* -** - Este algoritmo atribui uma prioridade dinâmica inversamente proporcional ao seu *deadline* absoluto. Quanto menor for o *deadline* absoluto, maior é a sua prioridade. Este algoritmo minimiza o atraso de execução de todas as tarefas, distinguindo-se dos algoritmos anteriores que favorecem a tarefa com maior prioridade durante toda a execução do sistema.

2.1.3 Acesso a Recursos Partilhados

Caso exista execução de tarefas concorrentes que partilhem recursos, o acesso a estes deverá ser em regime de exclusão mútua a fim de assegurar a sua integridade e o bom funcionamento das próprias tarefas. Sendo assim, devido ao facto de as tarefas correrem no acesso com prioridades diferentes, os sistemas de tempo real são também afectados pelo

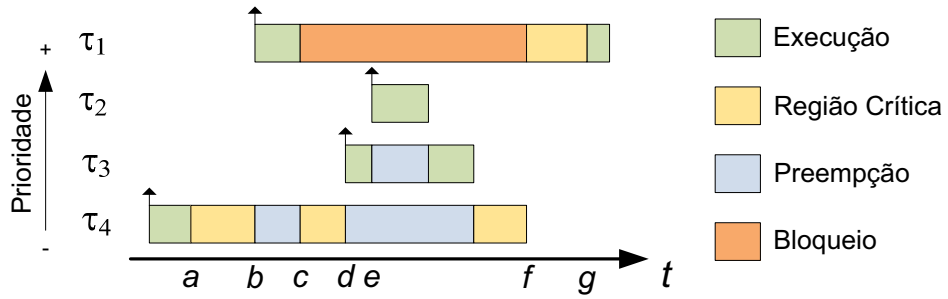


Figura 2.4: Demonstração do problema da inversão de prioridades.

problema da **inversão de prioridades**. Na figura 2.4, podemos observar como o problema da inversão de prioridades se coloca. As tarefas τ_1 e τ_3 partilham um dado recurso partilhado (indicado a azul), sem no entanto ser usado pela tarefa τ_2.

Como podemos observar, a tarefa τ_1 começa por aceder ao recurso partilhado, existindo depois a sua preempção por τ_3. No entanto, quando τ_3 tenta aceder ao recurso partilhado, verifica que este se encontra bloqueado, passando então para o estado bloqueado, passando a ser executada a tarefa τ_3. De seguida, a tarefa τ_2 causa preempção à tarefa τ_1 e é aqui que se verifica efectivamente o problema de inversão de prioridades! A tarefa τ_2 está a causar o bloqueamento da tarefa τ_3 através da preempção da tarefa τ_1, apesar de nunca utilizar o recurso partilhado.

No entanto, o problema da inversão de prioridades, é um fenómeno que não é possível ser evitado na sua totalidade nos sistemas de tempo-real. No entanto, podemos limitar e diminuir a sua duração, através de técnicas de sincronização no acesso aos recursos partilhados.

2.1.3.1 Técnicas Básicas de Sincronização

Inibição de Interrupções

A inibição de Interrupções, consiste em desactivar as interrupções do sistema, evitando assim a execução da rotina de interrupção que por si procede à activação de novas tarefas. Sendo assim, ao inibir-se as interrupções, impede-se também a activação de novas tarefas. No entanto, isto tem um lado extremamente negativo, sendo que a rotina de interrupção, na maioria dos sistemas de tempo-real, trata também de outros aspectos internos, como a contagem de tempo e a detecção de interrupções externas. Existe também o problema de tarefas de alta prioridade serem impedidas de executar devido à execução de tarefas de prioridade inferior

Resumindo, a inibição de interrupções garante a exclusão mutua no acesso aos recursos, no entanto a um preço elevado, como a perda de precisão temporal, perda de eventos externos do sistema e ainda o bloqueio de todas as tarefas por parte das tarefas de baixa prioridade.

Inibição de Preempção

A inibição de Preempção, consiste em impedir que uma tarefa em execução seja interrompida por outra tarefa de prioridade superior. Este método garante a exclusão mutua dos recursos partilhados e é uma forma de sincronização melhorada em relação à inibição

de interrupções. Interrupções do sistema continuam, mantendo-se a contagem temporal e a detecção de eventos externos. No entanto, o problema do bloqueio de todas as tarefas de alta prioridade por tarefas de baixa prioridade mantém-se.

2.1.3.2 Técnicas de Sincronização Baseadas em Semáforos

As técnicas de sincronização baseadas em semáforos, são técnicas simples que visam resolver os problemas existentes na utilização da inibição de interrupções e preempção, resolvendo ainda o problema da inversão de prioridades.

Existem várias técnicas de sincronização utilizando semáforos, no entanto só uma irá ser apresentada, sendo esta a utilizada no âmbito da dissertação. As técnicas de sincronização por semáforos mais conhecidas são:

- **Protocolo de Herança de Prioridades** - Detalhes podem ser encontrados em [SRL90]
- **Protocolo de Tecto de Prioridades** - Detalhes podem ser encontrados em [SRL90]
- **Politica de Pilha de Recursos** - Será descrito em baixo, mas poderão ser encontrados mais detalhes em [Bak91]

Politica de Pilha de Recursos (*Stack Resource Policy*)

A politica de pilha de recursos, é um protocolo estruturado para poder ser utilizado em sistemas de tempo-real cuja prioridade poderá ser fixa ou dinâmica. Este protocolo permite bloquear uma tarefa caso os recursos de que esta necessita, não estejam disponíveis no momento de activação. Neste protocolo, cada tarefa tem um nível de preempção, sendo

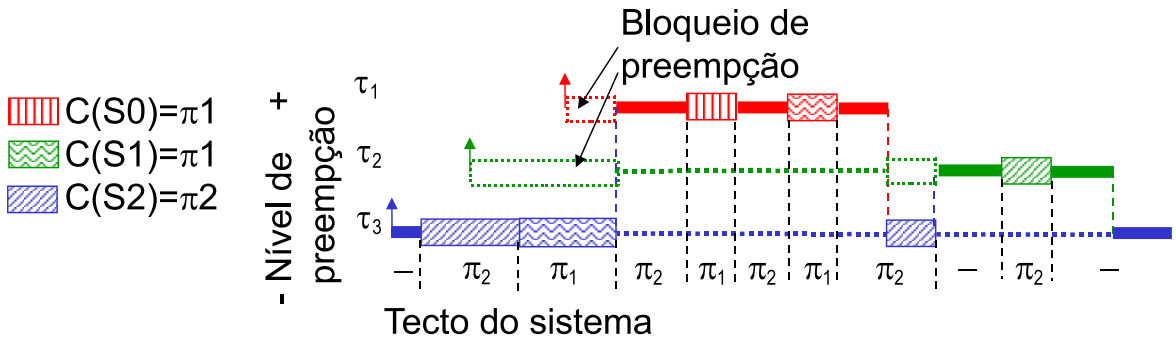


Figura 2.5: Exemplo do funcionamento da Politica de Pilha de Recursos.[LA08]

igual à prioridade no caso do algoritmo de escalonamento ser fixo, ou então, sendo inversamente proporcional ao *deadline* relativo no caso do algoritmo de escalonamento ser dinâmico. Quando a tarefa com maior prioridade é seleccionada pelo escalonador do sistema, é feita a comparação do seu nível de preempção com o tecto do sistema. Caso o nível da tarefa seja superior ao nível do tecto do sistema, a tarefa será colocada em execução, caso contrário, será bloqueada ficando a aguardar execução.

O protocolo determina que o tecto do sistema, é definido pelo valor máximo de preempção, do conjunto de semáforos que estejam fechados nesse dado momento. Sendo assim, cada

semáforo tem de forma independente, um nível de preempção, que é o valor máximo de preempção, do conjunto de tarefas a si associadas.

Sendo assim, só as tarefas com um nível de preempção superior ao do tecto de sistema é que serão executadas, pois isso significa que os recursos que irá utilizar estão disponíveis, caso contrário, os semáforos associados aos recursos, estariam fechados e consequentemente o tecto do sistema seria igual ao nível de preempção da tarefa, que neste caso seria bloqueada.

Esta politica de gestão de recursos, limita o problema de inversão de prioridades, impede o bloqueio indistinto das tarefas de alta prioridade por tarefas de baixa prioridade e ainda, respeita a temporização do sistema mantendo também o seu determinismo.

2.2 Executivos de Tempo-Real

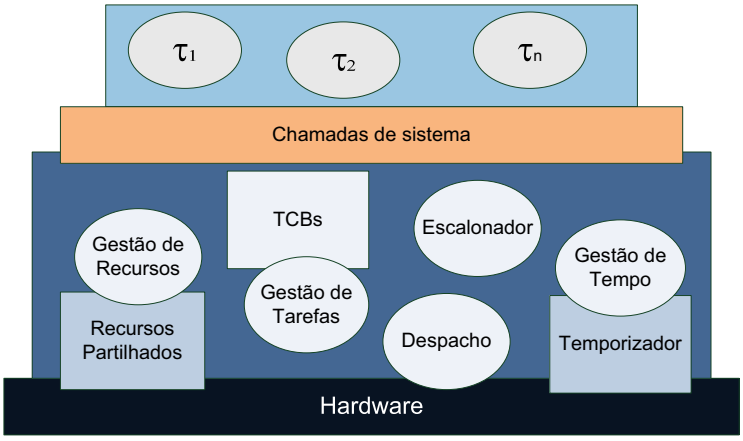


Figura 2.6: Representação da Arquitectura Interna de um Executivo de Tempo-Real.

Um executivo de tempo-real (figura 2.6), não é mais que o gestor de actividades do sistema de tempo-real. Responsável pela gestão da execução das tarefas, a sua correcta implementação é essencial para o bom funcionamento de todo o sistema. O executivo OReK[ASRdO07][dS08], serve de exemplo para demonstração de uma implementação de um executivo de tempo-real. Detalhes e pormenores sobre este executivo poderão ser encontrados no anexo A.

A existência de um executivo de tempo-real, permite que as diversas aplicações críticas possam ser construídas, utilizado os serviços e recursos disponibilizados. São normalmente de construção e arquitectura simples, conseguindo assim um elevado determinismo e também um desempenho aceitável.

Sendo assim, existe uma separação entre o desenvolvimento do executivo de tempo-real e das aplicações para sistemas de tempo-real. Isto permite que quem desenvolve as aplicações, não se tenha que preocupar com aspectos de gestão dos recursos, execução correcta das aplicações ou funcionamento dos aspectos internos relativamente a interrupções externas.

Assim, existe um conjunto de funcionalidades que se encontra presente na grande maioria dos executivos de tempo-real:

- **Temporizador**

- **Gestão de Interrupções Externas**
- **Gestor de Tarefas**
- **Agendador de Tarefas**
- **Escalonador de Tarefas**
- **Gestor de Recursos**
- **Chamadas ao Sistema**

2.2.1 Temporizador

Geralmente, o temporizador é constituído por uma rotina executada de forma periódica e contínua, em que o seu período de execução pode ser pré-programado, definindo assim a resolução de todo o sistema. A sua finalidade pode variar, desde controlar a activação periódica de tarefas, verificar o cumprimento de restrições temporais, medir intervalos de tempo, execução periódica do escalonador de tarefas, e entre outros.

2.2.2 Gestor de Interrupções Externas

A sua presença, permite fazer uma gestão adequada das interrupções externas. Geralmente está implementado sob a forma de uma rotina, cuja execução é feita cada vez que existir uma alteração do estado de um sinal de interrupção externa do sistema, que normalmente, estão ligados a sensores, botões ou outros tipos de detectores de eventos do ambiente que envolve o sistema.

A execução desta rotina permite verificar que tipo de interrupção foi gerada e consequentemente, poderá ser agendada para execução, uma tarefa esporádica pré-associada.

2.2.3 Escalonamento e Agendamento de Tarefas

O executivo de tempo-real, utiliza um escalonador, para poder determinar qual a tarefa com mais prioridade num determinado momento temporal do sistema. O escalonador, baseia-se nos algoritmos já descritos na secção anterior, para determinar as prioridades das tarefas.

Estes sub-sistemas são implementados sob forma de rotinas que são executadas, a cada período pré-determinado pelo temporizador.

2.2.4 Gestor de Recursos

O gestor de recursos, permite a um executivo de tempo-real, a estabelecer a exclusão mútua no acesso a recursos partilhados do sistema. Estes recursos podem ser periféricos do sistema, segmentos de memória partilhada ou um outro qualquer recurso, cujo acesso necessite de ser feito de forma exclusiva.

Os protocolos utilizados por este gestor, foram referidos na secção anterior.

2.2.5 Chamadas ao Sistema

Este poderá ser um dos sub-sistemas mais importante do executivo. Permite fazer a ligação das tarefas com o próprio executivo, por forma a definir uma interface de acesso entre as suas funcionalidades e as tarefas, permitindo ao programador construir e desenvolver

aplicações portáveis, que ao invés de dependerem da arquitectura do sistema, dependem apenas da presença do executivo.

2.3 Trabalho Relacionado

Como já foi explicado na introdução deste capítulo, nesta secção são apresentados diversos trabalhos executados para sistemas de tempo-real e sistemas reconfiguráveis.

Durante a projecção e construção de um sistema de tempo-real, existem vários problemas aos quais se deverá estar atento, para que durante a execução das suas funções, esteja apto a cumprir as tarefas que lhe são atribuídas.

Na secção 2.1 foi explicado que, para gerir as tarefas a executar, é utilizado um executivo de tempo-real que consumindo algum do tempo de processamento disponível, este ocupa-se de executar as suas funções como gestor do sistema. Este tempo ocupado pelo executivo, conhecido por *overhead*, caso seja em grande quantidade poderá afectar o desempenho das tarefas, devido a não existir tempo de processamento disponível para a execução das mesmas, correndo assim o risco de existir a perda do *deadline*. A possibilidade de executar as rotinas de gestão internas do executivo em *hardware* (escalonamento e agendamento para execução das tarefas, atendimento de interrupções e mudança de contexto) permite reduzir o *overhead* do sistema e consequentemente, reduzir ou até mesmo eliminar, possíveis ocorrências de perdas de *dealine*.

No entanto, as tarefas a gerir por um executivo, poderão não ser executadas em tempo útil devido às limitações do próprio processador do sistema (frequência de funcionamento, precisão aritmética, acesso aos recursos, entre outras). Neste caso, a implementação destas tarefas em hardware, torna-se uma forma de contornar o problema desde que exista a presença de um circuito reconfigurável com recursos disponíveis em número suficiente.

A execução de tarefas que possam partilhar entre si, recursos do sistema, apresentam problemas que podem surgir caso não sejam tomados certos cuidados, sendo um desses problemas a ocorrência de *deadlocks*[CES71]. Como este problema, é na realidade um problema de implementação das tarefas, caso não seja possível analisar detalhadamente o código fonte de uma dada tarefa, é necessário encontrar meios para detectar e evitar *deadlocks* entre tarefas. No entanto, uma possível implementação deste detector, teria que ser feito ao nível do executivo, podendo assim aumentar o *overhead* do mesmo, levando ao problema já explicado anteriormente. A implementação deste controlador em hardware, viria resolver ambos os problemas.

Soluções para estes problemas, provenientes de trabalhos executados por outros autores, são agora apresentadas. Também é apresentada, uma comparação entre um sistema de implementação exclusivamente em software um sistema com uma implementação mista.

2.3.1 Redução de *overhead* utilizando circuitos reconfiguráveis

Em [SHC07], o autor descreve como se pode através da aceleração por hardware reconfigurável, optimizar o atendimento de interrupções, mudança de contexto e agendamento de tarefas para um sistema de tempo-real.

Para este propósito, em vez de os procedimentos acima indicados, serem executados tradicionalmente pelo CPU, é feita a divisão do executivo em duas componentes, uma parte

implementada em software e outra parte implementada em hardware reconfigurável. Neste conjunto, o bloco do executivo implementado em software controla o bloco implementado em hardware, através do envio e recepção de sinais de controlo.

Os autores concluíram que a utilização de hardware reconfigurável, aumentou de forma significativa o desempenho do executivo, sendo também benéfico para a utilização do CPU, devido ao tempo de processamento ganho devido à redução do *overhead* do executivo. Os autores propuseram também, de uma forma optimizada, a implementação de um processador reconfigurável juntamente com o executivo de tempo-real, o que vai de encontro em parte, ao tema desta dissertação.

2.3.2 Arquitectura de um Co-Processador Dinâmico

Em [MGBN06], é feita a demonstração em como se pode ultrapassar as dificuldades, de implementar múltiplos co-processadores ou tarefas em hardware reconfigurável. O autor descreve como pode ser complicado, implementar ou configurar múltiplos co-processadores ou outros tipos de IP's devido às restrições na quantidade de blocos lógicos disponíveis e também, na forma como estes são interligados com o processador central.

Sendo assim, devido às FPGA's permitirem que apenas uma parte do seu circuito possa ser reconfigurado, é implementado um co-processador dinâmico. Este co-processador, tem um interface próprio sendo que o seu circuito, irá ser reconfigurado à medida que é necessário à execução das diversas tarefas do sistema operativo.

Para a reconfiguração, é utilizada uma ferramenta chamada *ROCCC-(Riverside Optimizing Compiler for Configurable Computing)*[UCR] que faz a compilação de código C para código VHDL. A ferramenta é usada para que as tarefas em C, possam ser transformadas em circuitos lógicos e assim, reconfigurando o co-processador dinâmico, para que seja este a executar a tarefa. O artigo descreve os detalhes desta ferramenta.

Sendo assim, demonstra-se como é possível ter múltiplas tarefas a executarem em hardware, obtendo um maior desempenho em relação à sua versão em software, evitando a exaustão dos recursos da FPGA.

2.3.3 Analisador de execução em hardware

Em [SVCG99] , o autor descreve como, disponibilizando um sistema operativo com capacidade reflectiva, se pode fornecer às aplicações para sistemas de tempo real, a possibilidade de analisarem a sua execução e consequentemente, alterarem as suas políticas de gestão de recursos do sistema.

As aplicações, têm ainda acesso a recursos remotos, considerando uma estrutura distribuída dentro do sistema tempo-real (sistemas heterogéneos).

Um dos pontos fundamentais, é o facto do sistema operativo estar implementado numa plataforma dinâmica, possibilitando o acesso às aplicações de funções implementadas directamente no hardware, sendo assim possível, acelerar diversas primitivas do sistema tempo-real e funções específicas das aplicações.

2.3.4 Escalonadores em Hardware

Em [CYC+05], o autor descreve duas implementações de um escalonador de tarefas, uma versão centralizada e uma dinâmica.

A estratégia centralizada, implementa o escalonador de tarefas por meio de uma tarefa que é executada num processador dedicado, ou então, uma implementação em hardware cujos processadores ou micro-controladores, poderão aceder através de uma interface. Esta estratégia apesar de ter um custo de área lógica menor no entanto, tem um maior custo na sua execução, devido ao facto de ser necessário sincronizar os acessos feitos pelos processadores.

A estratégia distribuída, consiste em ter um escalonador dedicado para cada processador. Tem um custo muito menor na sua execução devido ao facto de a troca de informação ser feita entre processadores, no entanto, o seu custo de área lógica é muito superior.

As análises sintéticas, demonstram que uma estratégia distribuída, devido ao facto dos escalonadores serem executados em paralelo, existe um claramente um menor custo na sua execução, sendo o seu pior caso (custo de execução mais *overhead*) inferior ao melhor caso da arquitectura centralizada (apenas o custo da execução).

Ainda falando em escalonadores implementados em hardware, em [SVCG99], o autor descreve uma implementação de um algoritmo de escalonamento e agendamento de tarefas, implementado em VHDL.

Esta implementação, permite retirar da componente de software, todo o tratamento de temporização e escalonamento das tarefas, limitando as interrupções apenas para quando é necessário fazer uma troca de contexto. São utilizadas filas de prioridades dinâmicas, cujas prioridades são actualizadas a cada clock do sistema.

Em ambos os artigos, demonstra-se os ganhos efectivos que se obtém, ao ser feita a implementação dos escalonadores em hardware. Novamente, acentua-se a motivação para a execução desta dissertação.

2.3.5 Comutação de tarefas entre processador genérico e circuito reconfigurável

Em [DWX+05], o autor descreve de forma breve um executivo de tempo-real, cujas tarefas podem ser implementadas em software ou em hardware. As aplicações a serem implementadas em hardware, segundo os autores, são escolhidas tendo em conta as restrições e condições, implementando as restantes em software.

Para fornecer às tarefas um método de comunicação, os autores, implementaram um sistema de comunicação uniforme baseado em passagem de mensagens, sendo a sua troca, feita num formato comum.

Para implementar a mudança de contexto para as tarefas implementadas em hardware, é necessário extrair todos os bits de informação de estado do *read back bitstream*. No entanto, os autores indicam que se deverá evitar-se a preempção de tarefas implementadas em hardware, pois o *readback* leva 800ms e requer muita memória, algo pouco disponível na FPGA. Para as tarefas implementadas em software, é feito pelo método normal em software, copiando os registos para a stack.

Para terminar, os autores testaram a implementação do executivo e das suas tarefas, comparando com os resultados de vários exemplos do *Motorola Powerstone Benchmark Suite* e concluíram que a utilização deste executivo de tempo-real consegue melhorar de forma muito significativa o desempenho de um sistema embutido.

2.3.6 Comparação entre *MicroC/OS-II* e uma Unidade de Tempo-Real

Em [NA07] , os autores comparam uma Unidade de Tempo Real implementada em VHDL com um executivo de tempo-real conhecido, o *MicroC/OS-II*[Mic]. É feita a comparação da quantidade de memória utilizada e ainda da quantidade de lógica necessária para ambas as implementações, verificando-se que a Unidade de Tempo Real, necessita de menos de metade da memória em comparação com a implementação em software.

Para além da lógica utilizada, foi também feita uma comparação da implementação em várias plataformas, verificando-se que numa *StratixII*[Alt], obtém-se o menor consumo de recursos.

A análise é terminada, comprovando que um executivo de tempo-real com suporte de hardware reduz o custo de projecto, quando ajustando a implementação de funcionalidades, incluindo apenas o necessário para as aplicações. A implementação de um executivo de tempo-real com suporte de hardware aumenta a possibilidade da sua utilização.

2.3.7 Controlador para Detecção de *Deadlocks* em Hardware

Em [SS05], o autor apresenta um algoritmo que permite detectar e evitar *deadlocks* entre tarefas. O autor faz apenas a abordagem da implementação do controlador em hardware reconfigurável. A abordagem utilizada foi a construção de uma máquina de estados finitos, sendo cada estado deste autómato, uma das acções suportadas pelo controlador.

Testes feitos ao controlador, revelaram que a frequência máxima de funcionamento é de 53,14Mhz. Foram também criados 3 cenários de teste, cenário óptimo em que o número de recursos consegue satisfazer qualquer processo, cenário linear em que requer o processamento de cada nível do grafo de alocação de recursos e por fim, o cenário de pior-caso em que representa o pior caso na análise do grafo de alocação de recursos feita pelos algoritmos. Na comparação da versão software com a versão implementada em hardware, os resultados foram, para o caso óptimo, um aumento de 98,78%, para o caso linear, um aumento de 89,39%, e para o cenário de pior-caso, um aumento de 98,64%.

2.4 Co-Processadores para Executivos de Tempo-Real

Nesta secção, é apresentada uma tabela onde é feita a comparação entre vários co-processadores já existentes, para sistemas de tempo real.

Co-Processadores	**Temporização por Hardware**	**Gestão/Escalonamento de Tarefas**	**Gestor de Interrupções**	**Comutação Rápida de Contexto**	**Comunicação/Sincronização entre Tarefas**	**Acesso Determinístico a Recursos Partilhados**	**Detecção/Prevenção de *Deadlocks***	**Gestão de Memória**	**Suporte para Multi-Processadores**	**Sistema Híbrido (Sw/Hw)**	**Interface de Programação**	**Tipo e Tecnologia de Implementação**	**Referência**
Plataforma *Hthreads*	✓	✓			✓					✓	✓	FPGA	[WA06]
Executivo de tempo-real sintetizado para hardware a partir de código em linguagem C	✓	✓	✓							✓	✓	ASIC	[CRL06]
Descrição da arquitectura de um processador SMT com escalonamento das tarefas implementado em hardware	✓	✓	✓						✓			Sim	[Met05]
Descrição do co-projecto de hardware-software do executivo de tempo-real do projecto Hthreads	✓	✓			✓					✓	✓	FPGA	[APA+05]
Análise de desempenho do RTOS $\mu C/OS - II$ com o módulo RTU	✓	✓	✓		✓				✓		✓	FPGA	[NL05]
Continua na próxima página													

Tabela 2.1: Tabela 1 de Co-Processadores para Sistemas de Tempo-Real

Co-Processadores	Temporização por Hardware	Gestão/Escalonamento de Tarefas	Gestor de Interrupções	Comutação Rápida de Contexto	Comunicação/Sincronização entre Tarefas	Acesso Determinístico a Recursos Partilhados	Detecção/Prevenção de *Deadlocks*	Gestão de Memória	Suporte para Multi-Processadores	Sistema Híbrido (Sw/Hw)	Interface de Programação	Tipo e Tecnologia de Implementação	Referência
Descrição da *µFramework* para co-projecto de hard/software de SO's com suporte para multi-cpu	✓	✓	✓			✓		✓	✓	✓	✓	FPGA	[MI05]
Descrição resumida do módulo Sierra 16 comercializado pela empresa RealFast e baseado na RTU	✓	✓	✓		✓				✓		✓	FPGA	[Rea05]
Sistema operativo implementado em hardware para gestão de sistemas embutidos reconfiguráveis dinamicamente baseado num processador, co-processador e blocos de hardware configuráveis		✓			✓					✓	✓	FPGA	[BJS04]
Continua na próxima página													

Tabela 2.2: Tabela 2 de Co-Processadores para Sistemas de Tempo-Real

Co-Processadores	Temporização por Hardware	Gestão/Escalonamento de Tarefas	Gestor de Interrupções	Comutação Rápida de Contexto	Comunicação/Sincronização entre Tarefas	Acesso Determinístico a Recursos Partilhados	Detecção/Prevenção de *Deadlocks*	Gestão de Memória	Suporte para Multi-Processadores	Sistema Híbrido (Sw/Hw)	Interface de Programação	Tipo e Tecnologia de Implementação	Referência
Plataforma multi-tarefa baseada num sistema integrado reconfigurável heterogéneo com suporte em hardware para o escalonamento e comunicação entre tarefas		✓			✓				✓			FPGA	[MNM+04]
Arquitectura RTP e sua implementação em hardware através de um escalonador e uma matriz para gerir a atribuição dos recursos (processadores, dispositivos) às tarefas	✓	✓	✓		✓				✓			FPGA	[IW04], [YW04]
Modulo ARPA-OSC para o processador ARPA-MT	✓	✓			✓					✓	✓	FPGA	[ASRdO07]

Tabela 2.3: Tabela 3 de Co-Processadores para Sistemas de Tempo-Real

2.5 Contributos

Através do desenvolvimento dissertação, pretende-se dar um contributo à área de desenvolvimento de co-processadores para sistemas de tempo-real. Sendo assim, este projecto apresenta algumas inovações que são agora realçadas.

- Suporte de processadores genéricos, por parte do co-processador.
  - Permite que sejam usados todos os tipos de processadores, desde que tenham a capacidade de receber pedidos de interrupção externos.
- Implementação de operações de execução previsível e determinística.
- Gestão paralela da execução e activação das tarefas do sistema.
- Controlo temporal dos pedidos de interrupção provenientes de fontes externas.

Capítulo 3

Desenvolvimento do Co-Processador

Neste capítulo, irá ser apresentada a arquitectura do co-processador. Na introdução, é feita a comparação entre um sistema tradicional implementado unicamente em *software*, com um sistema híbrido composto por uma componente em *software* e uma componente em *hardware*, representado por um co-processador. O objectivo desta comparação, é demonstrar como a utilização de um co-processador se torna benéfica para um sistema de tempo-real, ao melhorar o seu desempenho e determinismo. De seguida, é feita a descrição da arquitectura do co-processador, sendo explicado com detalhe, o funcionamento de cada módulo interno. As descrições são auxiliadas por diagramas que procuram ajudar à compreensão da arquitectura do co-processador e também do seu funcionamento.

A arquitectura desde co-processador, permite-lhe suportar as seguintes capacidades:

- Integração com processadores genéricos que suportem interrupções externas.
- Acesso às funcionalidades internas, utilizando instruções atómicas.
- Gestão de pedidos de interrupção externas.
- Gestão da execução de tarefas (Periódicas, Aperiódicas e Ordinárias).
- Gestão controlada do acesso a recursos partilhados.

Todas estas capacidades, serão explicadas de forma detalhada ao longo deste capítulo, sendo a abordagem feita em cada módulo apropriado.

3.1 Introdução

Implementações de executivos de tempo-real, quando implementados com o auxílio de um co-processador, a sua forma de execução muda radicalmente. Isto acontece devido á capacidade de um co-processador, poder paralelizar as diversas funcionalidades existentes num executivo, conseguindo assim, execução de acções sobre as tarefas de forma paralela.

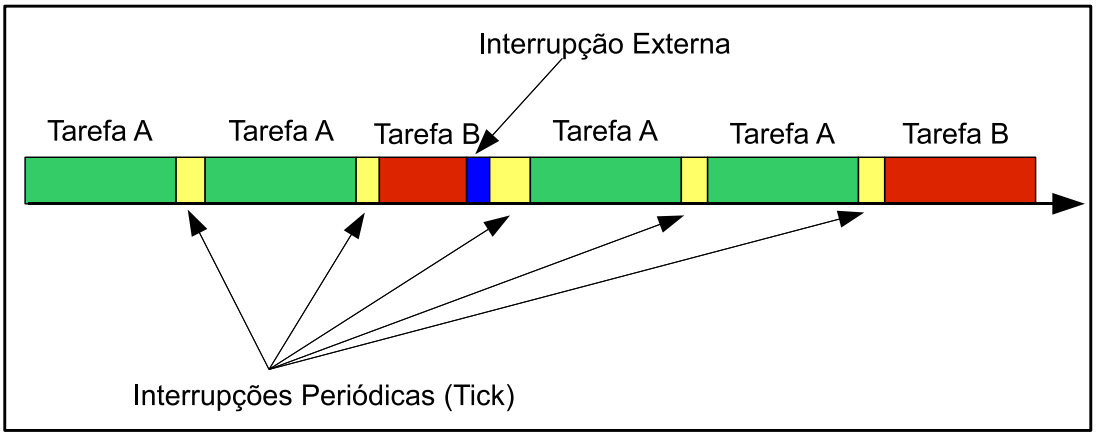
Na figura 3.1(a), podemos observar um executivo cuja implementação é exclusivamente feita em software. Neste caso, cada vez que é necessário executar rotinas internas para a gestão temporal das tarefas, é necessário interromper o processamento da tarefa que se encontra no contexto do processador, trocando com o contexto do executivo, desperdiçando tempo de processamento.

Para além deste problema, existem também o problema das tarefas esporádicas ou aperiódicas. Estas tarefas são executadas segundo pedidos provenientes das linhas de

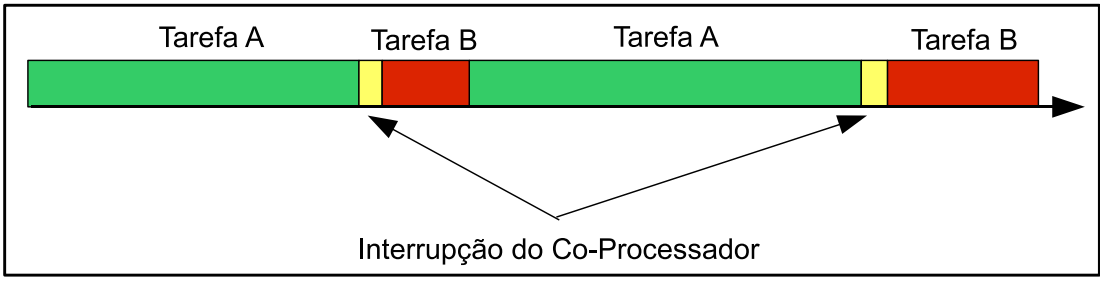
interrupção externas, e caso exista uma grande quantidade de pedidos, o processador poderá ser *inundado* com interrupções, podendo mesmo inibir a gestão temporal das tarefas, provocando atrasos temporais na activação de tarefas periódicas, quebrando assim a sincronização temporal, criando até mesmo situações nefastas idênticas à perda de *deadlines* e ainda, introdução de indeterminismo. Pode-se também referir que existe ao mesmo tempo, uma clara perda de desempenho.

Rotinas programadas para gestão temporal das tarefas, gestão das interrupções externas e escalonamento de tarefas, quando executadas de forma convencional, não podem tirar proveito de o seu código ser executado em paralelo, como por exemplo, gerir o tempo de cada tarefa de forma paralela, gerir o tempo mínimo de activação para cada linha de interrupção e até mesmo, tirar proveito de um escalonamento de tarefas paralelo. O serviço de atendimento de pedidos de interrupção externos, fica por esta razão prejudicado, pois não tem capacidade para atendimento de pedidos de forma paralela. Isto prejudica o desempenho do sistema, devido à necessidade de atender os pedidos de forma sequencial, correndo-se ainda o risco de outros pedidos serem perdidos devido à inibição de interrupções imposta pela rotina do serviço de atendimento de interrupção.

Dependendo da forma como é feita a gestão do estado das tarefas, as rotinas de gestão das tarefas, tendem também a estar extremamente dependentes do número de tarefas existentes no sistema, podendo o seu tempo de execução variar consideravelmente, sendo este aspecto, uma causa da inserção de algum indeterminismo no sistema.



(a) Execução do executivo implementado unicamente em software.

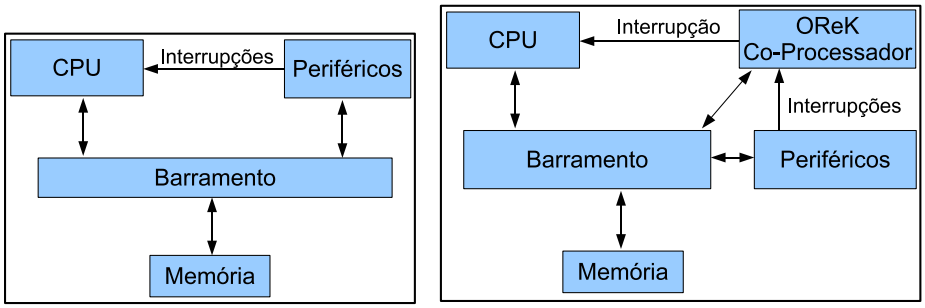


(b) Execução do executivo com co-processador.

Figura 3.1: Executivo implementado em Software contra implementação com co-processador.

Observe-se então agora, a figura 3.1(b), que representa uma implementação com o auxilio de um co-processador. Como passa a ser o co-processador que trata da gestão das tarefas e da recepção de pedidos de activação externos, o processador é apenas interrompido, quando efectivamente é necessário fazer uma troca do contexto de tarefas, evitando-se o desperdício do tempo de processamento.

Passa também, a ser possível implementar as operações em hardware aproveitando assim a possibilidade de se explorar o paralelismo, podendo por exemplo, o algoritmo de escalonamento (secção 3.2.5.3) de tarefas passar a ser implementado em paralelo, resultado de uma execução muito mais rápida e precisa. Todas as operações do co-processador, passam também a ter um tempo de execução fixo, melhorando assim o determinismo do sistema.



(a) Sistema Embutido Tradicional. (b) Sistema Embutido com Co-Processador.

Figura 3.2: Comparação de Arquitecturas entre dois sistemas embutidos distintos.

Observando as figuras 3.2(a) e 3.2(b), pode-se observar as diferenças entre um sistema embutido tradicional e um sistema embutido com co-processador. Um sistema embutido tradicional, tem o processador ligado a um barramento local por forma a aceder aos periféricos disponíveis, sendo que os periféricos que sinais de interrupção, têm estes ligados directamente ao processador.

No entanto, quando o co-processador está presente, as interrupções dos periféricos são ligadas ao co-processador, para que possam ser analisadas e controladas, sendo depois o co-processador ligado à entrada de interrupções externas do processador do sistema. O controlo do co-processador e do resto dos periféricos é feito pela forma tradicional, acedendo-lhes através do barramento local.

No contexto deste trabalho e para efeitos de teste e análise, foi utilizado o *Processsor Local Bus* versão 4.6 da Xilinx [Xil07], estando os efeitos da sua utilização descritos no capítulo 4.

3.2 Arquitectura do Co-Processador

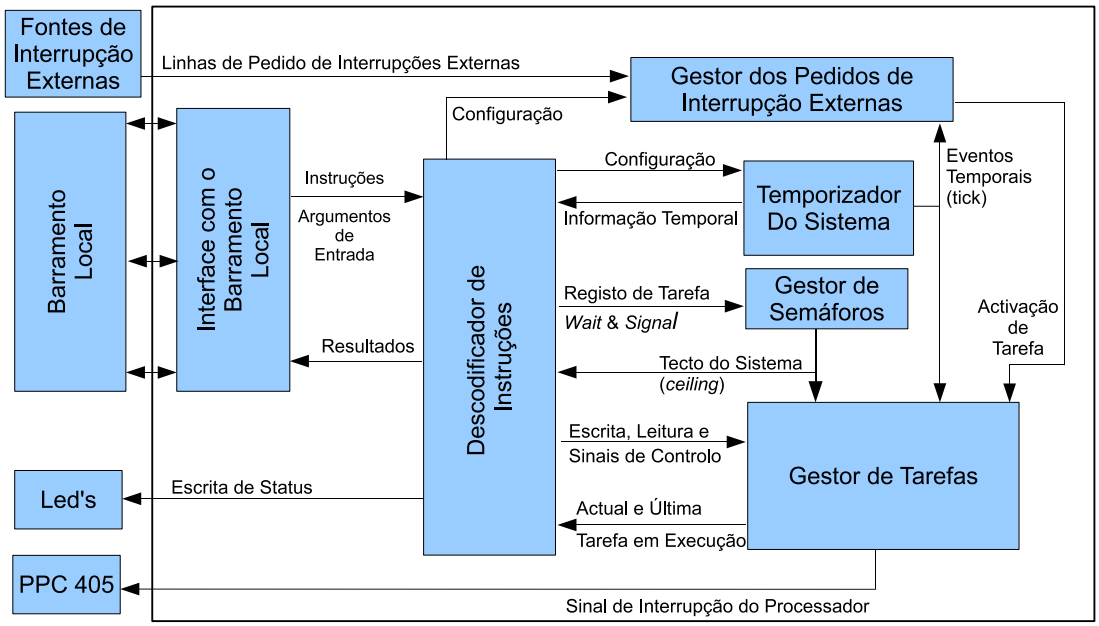


Figura 3.3: Diagrama de Implementação do Co-Processador.

Na figura 3.3 podemos observar a constituição interna do co-processador e as principais ligações entre os blocos. O co-processador é constituído pelos seguintes blocos:

- **Interface com o Barramento Local**
  - Interface de ligação entre o Descodificador de Instruções e o Barramento Local, que contém os registos do co-processador.
- **Descodificador de Instruções**
  - Descodifica as instruções, escritas num registo específico do Co-Processador.
- **Temporizador do Sistema**
  - Temporizador de 64 bits programável do co-processador.
- **Gestor dos Pedidos de Interrupção Externas**
  - Gere os pedidos de interrupções provenientes de periféricos externos.
- **Gestor de Tarefas**
  - Guarda os atributos temporais das tarefas gerindo ainda, a transição entre estados, implementa o escalonador de tarefas, sendo responsável pelo pedido de interrupção do co-processador.
- **Controlador de Semáforos**
  - Auxilia na gestão de recursos partilhados.

Nas próximas sub-secções, a arquitectura de cada bloco será explicada em detalhe, dando atenção também à sua forma de funcionamento e configuração.

3.2.1 Interface com o Barramento Local

Nesta secção, é apresenta-se a forma de comunicação com o co-processador. Utilizando registos de leitura e escrita, estabelece-se a interacção entre o processador genérico do sistema e o co-processador, sendo assim possível, configurar e gerir o seu funcionamento.

Existem três tipos de registos implementados no interface de ligação, sendo os registos de onde só é possível lêr informação (tabela 3.1), registos onde só é possível escrever (tabela 3.3) e por fim, os registos de entrada de argumentos e saída de resultados, para as operações do co-processador (tabela 3.2).

Os registos existentes no co-processador, são os seguintes:

Registos Especiais apenas de Leitura		
Macro de Software	**Localização**	**Conteúdo**
SystemError	Registo 0	Erro resultante da execução da última instrução.
TimerValue_HI	Registo 1	32 bits mais significativos do temporizador interno do co-processador.
TimerValue_LO	Registo 2	32 bits menos significativos do temporizador interno do co-processador.
TimerTickValue_HI	Registo 3	32 bits mais significativos do temporizador de ticks do co-processador.
TimerTickValue_LO	Registo 4	32 bits menos significativos do temporizador de ticks do co-processador.
TimerResolution	Registo 5	Resolução actual temporal do Co-Processador.
SystemCeiling	Registo 28	Valor actual do tecto do sistema (ver secção 3.2.6 para mais detalhes).
CurrentTask	Registo 29	Número da tarefa em actual execução.
LastTask	Registo 30	Número da última tarefa executada.
CoP_PreDefined_Values	Registo 31	**Em notação *Little Endian*:** bits 7 a 0 → nº máximo de Tarefas. bits 15 a 8 → nº máximo de Semáforos. bits 23 a 16 → nº máximo de Interrupções. bits 31 a 24 → reservado para futura utilização.

Tabela 3.1: Registos Especiais apenas de Leitura do Co-Processador.

Registos de Argumentos e Resultados		
Macro de Software	**Localização**	**Conteúdo**
Arg0 - Arg9	Registo 7 a 16	Argumentos de entrada das primitivas.
Res0 - Res8	Registo 19 a 27	Resultados obtidos pelas primitivas.

Tabela 3.2: Registos de Argumentos e Resultados do Co-Processador.

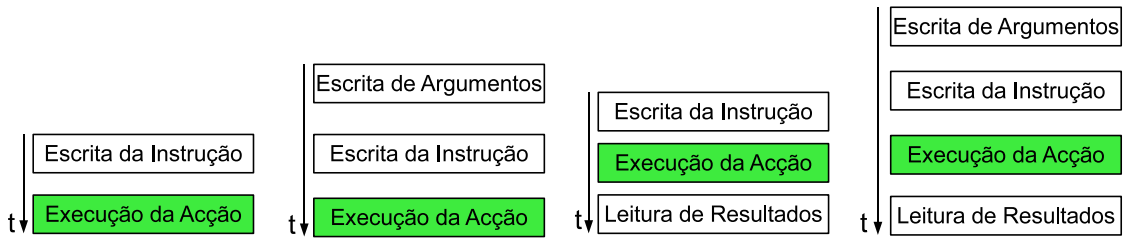
Registos Especiais apenas de Escrita		
Macro de Software	**Localização**	**Conteúdo**
Instruction	Registo 6	Instrução a executar.
InterruptionsControl	Registo 17	Inibição de interrupções: 1→ Interrupções inibidas. 0→ Interrupções activas.
Output Register	Registo 18	4 bits menos significativos ligados aos leds da FPGA (utilizado para testes apenas).

Tabela 3.3: Registos Especiais de Escrita(apenas) do Co-Processador.

3.2.2 Descodificador de Instruções

Para que fosse possível um co-processador genérico o suficiente, com o fim de poder ser utilizado em conjunção com o maior número de sistemas embutidos, era necessário utilizar um método de acesso que fosse suficientemente compatível e versátil. A arquitectura teria que suportar um meio de executar operações atómicas, que pudessem ter argumentos de entrada, sendo também possível ler os resultados obtidos da execução.

Assim, uma solução encontrada foi a utilização de registos endereçáveis (ver Secção 3.2.1) pelo processador, em que este poderia escrever argumentos, ordenar execução de instruções e ler os resultados obtidos.



(a) Instrução sem Argumentos (b) Instrução com Argumentos (c) Instrução com Resultados (d) Instrução com Argumentos e Resultados

Figura 3.4: Passos necessários para a execução de instruções.

Quando é necessário executar uma operação do co-processador, é escrito no registo de Instrução (ver tabela 3.1), o código correspondente a essa mesma primitiva. Ao existir uma alteração do registo de instrução, é feito um bloqueamento do barramento de comunicação do lado do co-processador, por forma a bloquear o processador e impedir que este continue a execução do programa, sem que a instrução do lado do co-processador, termine a sua execução. A aproximação de bloquear o acesso ao co-processador, foi escolhida devido à complexidade da arquitectura do co-processador e por ser uma solução eficaz e simples de gerir devidamente o acesso às suas funcionalidades, garantindo uma correcta execução de cada operação.

A figura 3.2.2, mostra os passos necessários a dar, para a execução das diversas instruções. Existem instruções sem argumentos (figura 3.4(a)), que basta simplesmente escrever no registo de instrução, o código correspondente. Existem instruções com argumentos (figura 3.4(b)), em que é necessário escrever primeiro os argumentos nos registos específicos (ver

tabela 3.2), sendo de seguida escrito o código da instrução a executar no registo apropriado. Existem também instruções apenas com resultados (figura 3.4(c)), em que se escreve o código da instrução, sendo depois lidos dos registos de saída (ver tabela 3.2), os resultados obtidos. Por último, existem também instruções com argumentos e que devolvem resultados (figura 3.4(d)), sendo necessário então, escrever os argumentos nos registos apropriados já indicados, escrever o código da instrução e ler de seguida, os resultados dos registos também apropriados e já indicados.

Dentro do descodificador de instruções, existe uma máquina de estados do tipo *Mealy* que monitoriza o sinal de aviso de execução, que ao ser colocado a 1, obriga a máquina de estados a ler o registo de instrução, e transitar para o estado correspondente.

É importante ainda dizer que, o descodificador suporta 6 tipos de instruções atómicas:

- Instruções para gestão temporal (detalhes descritos na tabela 3.4).
- Instruções para gestão do atendimento de pedidos de interrupção externos (detalhes descritos na tabela 3.5).
- Instruções para gestão das tarefas (detalhes descritos na tabela 3.6).
- Instruções para gestão dos semáforos (detalhes descritos na tabela 3.7).
- Instruções para gestão do executivo (detalhes descritos na tabela 3.8).
- Instrução de Terminação de Tarefa.

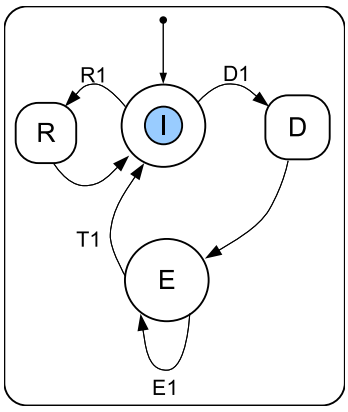


Figura 3.5: Máquina de estados do Descodificador de Instruções.

Na figura 3.5, está representada a máquina de estados do descodificador.

- Procedimentos
  - **R -** *Reset* do descodificador de Instruções
  - **D -** Descodificação de qual instrução deverá ser executada
- Estados
  - **I -** Estado inicial, de espera pela alteração do registo de instrução e transição para o estado de execução de uma tarefa.
  - **E -** Estado de execução de uma tarefa, permanece neste estado até a execução da instrução estar concluída.
- Transições (sincronizadas com o *Clock* do sistema)
  - **R1 -** *Reset* do descodificador foi ordenado.
  - **D1 -** Alteração do registo de instrução.
  - **E1 -** Execução da tarefa (instruções multi-ciclo).
  - **T1 -** Instrução terminou a sua execução (último ciclo de execução ou único para instruções de 1 ciclo).

3.2.2.1 Instruções Suportadas

Nesta secção são apresentadas as instruções suportadas pelo descodificador de instruções.

Instrução	Acção	# Ciclos
Set Timer Resolution	Configura a resolução do temporizador.	1
Timer Enable	Activa o Temporizador Interno do Co-Processador.	1
Timer Disable	Desactiva o Temporizador Interno do Co-Processador.	1
Timer Tick Enable	Activa o gerador de *Ticks* do Co-Processador.	1
Timer Tick Disable	Desactiva o gerador de *Ticks* do Co-Processador.	1
Timer Tick Reset	Reinicia o contador de *Ticks* do Co-Processador	2

Tabela 3.4: Instruções de gestão temporal.

Instrução	Acção	# Ciclos
Interrupt Control Setup	Configura a detecção dos pedidos de activação de tarefas provenientes de uma dada linha de interrupção externa (ver detalhes na Secção 3.2.4.1).	1
MIT Value Setup	Configura o Intervalo Mínimo entre activações, para uma dada linha de interrupção (ver detalhes na Secção 3.2.4.2).	1
Get Interrupt Value	Lê o valor da linha de interrupção externa, que provocou um pedido de activação de uma dada tarefa associada.	1

Tabela 3.5: Instruções de gestão dos pedidos de interrupção externos.

Instrução	Acção	# Ciclos
Register Task to Interrupt Line	Regista uma dada tarefa com uma dada linha de interrupção externa.	1
Get Task	Devolve os parâmetros temporais e de estado de uma dada tarefa.	2
Add Task	Adiciona uma tarefa ao co-processador.	3
Remove Task	Remove uma tarefa ao co-processador.	3
Start Task	Inicia a execução de uma Tarefa.	3
Stop Task	Ordena a Paragem de execução de uma Tarefa.	3
Activate Task	Obriga a passagem de uma tarefa do estado *Idle* para *Ready*.	3

Tabela 3.6: Instruções de gestão das tarefas.

Instrução	Acção	# Ciclos
Lock Semaphore	Bloqueia um semáforo (detalhes em 3.2.6).	2
Unlock Semaphore	Liberta um dado semáforo (detalhes em 3.2.6).	2
Reset Semaphore	Liberta um dado semáforo (detalhes em 3.2.6).	2
Register Semaphore	Adiciona uma tarefa ao co-processador.	3

Tabela 3.7: Instruções de gestão dos semáforos.

Instrução	Acção	# Ciclos
Set Algorithm	Configura o algoritmo usado no escalonamento de tarefas.	1
Enable Preemption	Activa a preempção de tarefas.	1
Disable Preemption	Desactiva a preempção de tarefas.	1

Tabela 3.8: Instruções especificas do executivo.

Instrução de Terminação de Tarefa

A instrução atómica de terminação de tarefa (*Terminate Task*), é uma instrução especial que difere do funcionamento normal em relação às outras instruções. Isto acontece devido ao facto do seu número de ciclos ser dependente do número de ciclos que o escalonador de tarefas demora a obter a tarefa com maior prioridade (o escalonador é abordado na secção 3.2.5.3). Quando a instrução de terminação de tarefa é executada, a instrução irá permanecer no estado de execução (ver figura 3.5), e a execução do código ficará bloqueada até o co-processador terminar a execução da instrução, sendo necessário esperar como foi indicado, pela terminação do escalonador de tarefas. Após o processador retomar a execução, poderá ir lêr aos registos do co-processador (ver tabela 3.1), qual a próxima tarefa a colocar no contexto de execução.

3.2.3 Temporizador do Sistema

Para fazer a gestão dos *ticks* do sistema, o co-processador implementa um módulo de gestão temporal (figura 3.6). Como o co-processador irá funcionar a uma frequência na ordem das centenas de Mhz, é fundamental que o módulo de controlo temporal tenha uma gama de representação suficientemente grande por forma a que a contagem temporal não entre em *overflow* durante o tempo útil de funcionamento do sistema embutido. Sendo assim, o gestor temporal está implementado com uma precisão de 64 bits.

Em paralelo com a contagem de tempo, o módulo de gestão temporal implementa também um *scaler*, que permite configurar a resolução temporal do co-processador. A resolução máxima é igual ao período do relógio do co-processador, no entanto, a utilização da resolução máxima não é praticável, devido ao facto do escalonador e *dispatcher* de tarefas, levarem alguns ciclos de relógio para obter um resultado e ainda, ser necessária uma mudança de contexto.

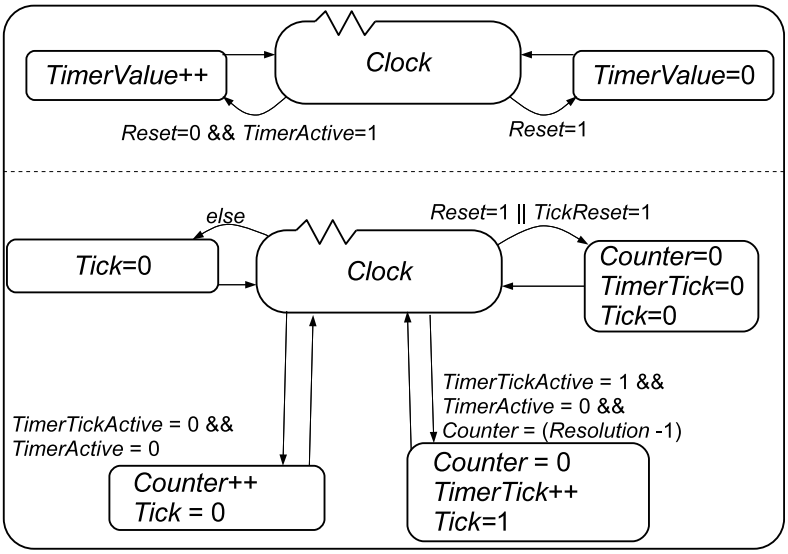


Figura 3.6: Modelo de estados do temporizador.

3.2.4 Gestor dos Pedidos de Interrupção Externas

O gestor dos pedidos de interrupção externas, pode ser considerado como uma das mais valias deste co-processador. Este gestor, tem a capacidade de detectar pedidos de interrupção em linhas externas, transformando-os em pedidos de activação de tarefas, instaurar um intervalo mínimo entre cada pedido e encaminhar os pedidos de activação para as suas tarefas correspondentes.

A instauração deste gestor, permite controlar os pedidos de interrupção proveniente de periféricos externos, de forma a controlar o impacto que estes têm sobre a execução das tarefas no processador do sistema, que caso não existisse qualquer controlo, seria a interrupção constante para atendimento de pedidos externos. Este caso foi explicado na secção 3.1 deste capítulo.

Na figura 3.7 está representado a arquitectura interna do gestor de tarefas, podendo-se observar três módulos internos:

- Conector de Linhas de Interrupção Externas (secção 3.2.4.1).
- Analisador Temporal do Intervalo Mínimo entre Activações (secção 3.2.4.2).
- Gestor de interligação entre as linhas de activação e as próprias tarefas (secção 3.2.4.3).

O funcionamento do gestor de interrupções externas, foi projectado para ser simples e eficiente, de forma a minimizar o número de ciclos de relógio necessários de processamento, desde o pedido de activação da tarefa por parte da linha externa, até à activação da própria tarefa.

O co-processador no acto da sintetize, necessita de saber quantas linhas de interrupção externas irá suportar, pois para cada linha, é gerado um conector de interrupção e um analisador temporal, com o intuito de paralelizar a análise e a activação de tarefas.

O funcionamento da gestão de interrupções é bastante simples, quando uma linha de interrupção externa, altera o seu estado de forma a sinalizar um pedido, o conector de interrupção de acordo com a sua configuração pré-definida *via software*, detecta a alteração de estado colocando o sinal de activação de tarefa a '1', indicando que se deseja activar uma

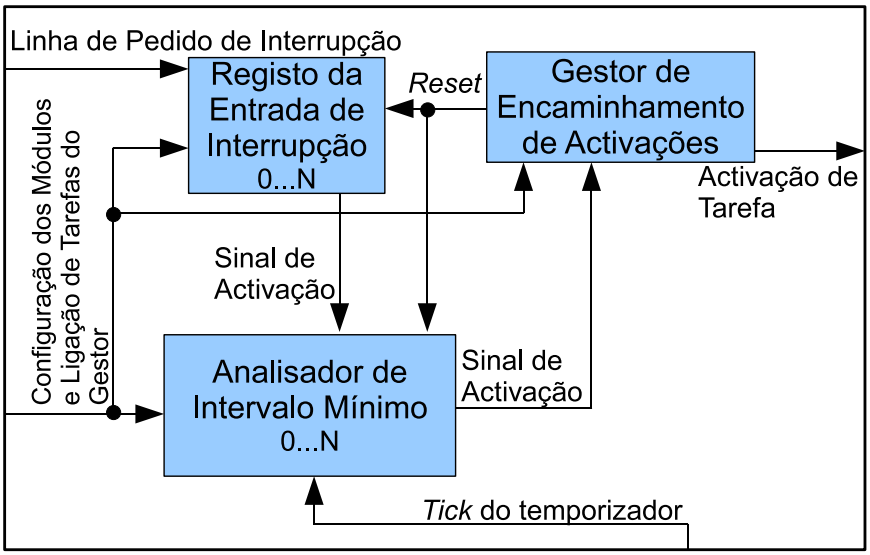


Figura 3.7: Modelo do Gestor de Interrupções Externas.

dada tarefa. De seguida, é feita a verificação pelo analisador temporal, se o tempo mínimo entre activações está a ser cumprido e caso esteja, o sinal de activação é encaminhado até ao gestor de interligação, que de acordo com a configuração pré-definida *via software*, encarrega-se de encaminhar o sinal de activação até à tarefa correspondente. Caso o tempo mínimo entre activações não tenha ainda sido cumprido, o sinal de activação não é encaminhado para o gestor, mantendo o seu valor e esperando que o intervalo de tempo atinja o seu limite.

O sinal de activação, ao ser enviado para a tarefa, em paralelo é também enviado um sinal de *reset* para o conector de interrupção e para o gestor temporal, com a finalidade de colocar o sinal de activação de tarefa a '0' e reiniciar a contagem do tempo mínimo entre activações. É também activado durante um ciclo de relógio, um sinal de pedido de execução do escalonador, para que este seja executado com a finalidade de, caso a tarefa activada tenha uma prioridade superior, seja imediatamente colocada no contexto do processador.

3.2.4.1 Registo de Entrada de Interrupção

O registo de entrada de interrupção (figura 3.8) é constituído por dois elementos, um *debouncer* e um detector para pedidos de activação.

Para que as linhas de interrupção pudessem ser utilizadas de uma forma genérica, é necessário considerar os casos em que os pedidos de activação de tarefas provenham de fontes instáveis. Um exemplo de uma fonte instável, são botões de pressão, que poderão causar instabilidade na linha de activação, fazendo alterar o seu valor de uma forma rápida e abrupta. Para estes casos, é necessário que o seu valor seja estabilizado para consequentemente, ser avaliado.

Sendo assim, para prevenir e evitar falhas ou detecções irregulares na detecção de interrupções, o conector de interrupções implementa um *debouncer*, para que seja possível estabilizar o valor na linha, tornando possível a análise de uma forma síncrona.

O detector dos pedidos de activação, é utilizado para, de acordo com a configuração pré-indicada, detectar alterações na linha proveniente do *debouncer* e verificar se existe uma

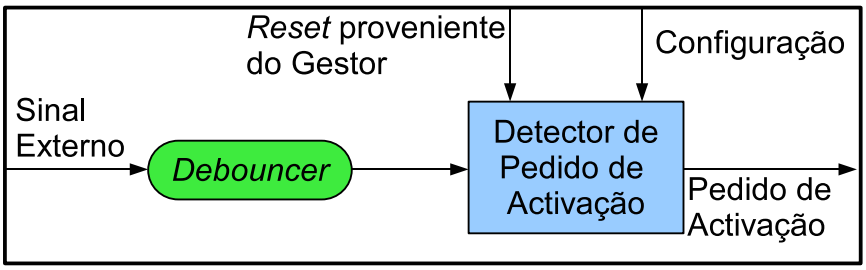


Figura 3.8: Modelo de um Conector de Interrupção.

alteração no sinal que corresponda efectivamente a um pedido de activação. Tal aconteça, a linha de saída dos pedidos de activação é colocada a '1'.

Debouncer

Observe-se a figura 3.9. O *debouncer* contém dois buffers internos, cujos valores são modificados a cada ciclo de relógio, sendo que o valor do buffer B1, é atribuído com o valor proveniente directamente da linha externa, sendo que o buffer B2, é atribuído com o valor actual do buffer B1. É ainda também feita a cada ciclo de relógio, a verificação lógica se efectivamente o sinal está estabilizado ou não. Para isso é usada a seguinte condição lógica $(\neg(B1 \oplus B2)) = 1$ e que caso a condição seja cumprida, então é atribuído ao sinal de saída do *debouncer* o valor do buffer B1.

Temporalmente, a utilização do *debouncer*, provoca um atraso de 3 ciclos de relógio na detecção do pedido de activação.

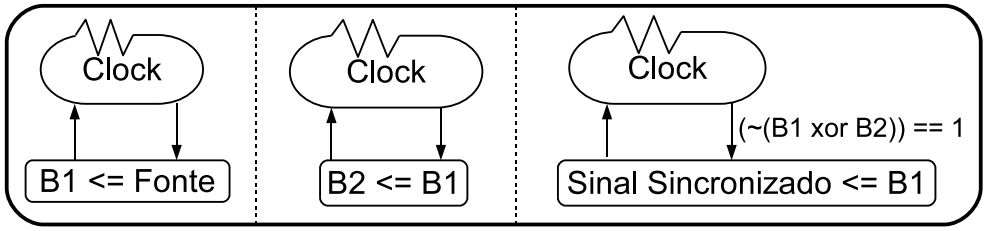


Figura 3.9: Modelo do *debouncer*.

Detector de pedidos de activação

O funcionamento do detector dos pedidos de activação, passa por fazer a comparação a cada ciclo de relógio do sinal proveniente do *debouncer* e compara-lo com o seu estado do último ciclo. Se os estados forem diferentes, é porque houve uma alteração e consequentemente é necessário verificar, se está presente um pedido de activação, segundo a configuração pré-programada. Caso esteja, o sinal de saída do connector de interrupções, para indicação de activação de uma dada tarefa, é colocado a '1'.

As seguintes configurações são possíveis da detecção dos pedidos de activação das tarefas.

- Sem detecção.
- Detecção na transição do sinal de 0→1.

- Detecção na transição do sinal de 1→0.
- Detecção na transição do sinal de 0→1 e 0→1.

3.2.4.2 Analisador de Intervalo Mínimo

O analisador de intervalo mínimo (figura 3.10), não é mais do que um controlador de encaminhamento do pedido de activação. O seu funcionamento, baseia-se na contagem do número de *ticks* do sistema, gerados pelo temporizador, de forma a verificar se o intervalo mínimo entre activações já foi cumprido.

De forma a não atrasar o sinal do pedido de activação, utiliza-se um *multiplexer* assíncrono. Este *multiplexer* liga à saída, uma das duas possíveis entradas, sendo a primeira o sinal do pedido de activação e a segunda, um sinal zero implícito.

O controlo do *multiplexer* é feito de acordo segundo a condição lógica

$$Mux = (Reset \wedge \neg MIT) \vee \neg Contador$$

e caso a condição seja verdadeira, o sinal do pedido de activação de entrada é imediatamente encaminhado para o sinal de saída do analisador temporal. No entanto, caso a condição seja falsa, é atribuído o valor de zero, mantendo-se assim caso exista, uma possível activação que exista, em espera.

Está presente também, um contador do tempo de intervalo mínimo. O contador começa no seu estado inicial, sendo-lhe atribuído o valor do intervalo mínimo (*MIT*), e por cada *tick* de sistema produzido pelo temporizador, o contador é decrementado em uma unidade de tempo. O contador ao atingir o valor zero, provoca a alteração da condição que controla o *multiplexer*, permitindo o encaminhamento do sinal de pedido de activação.

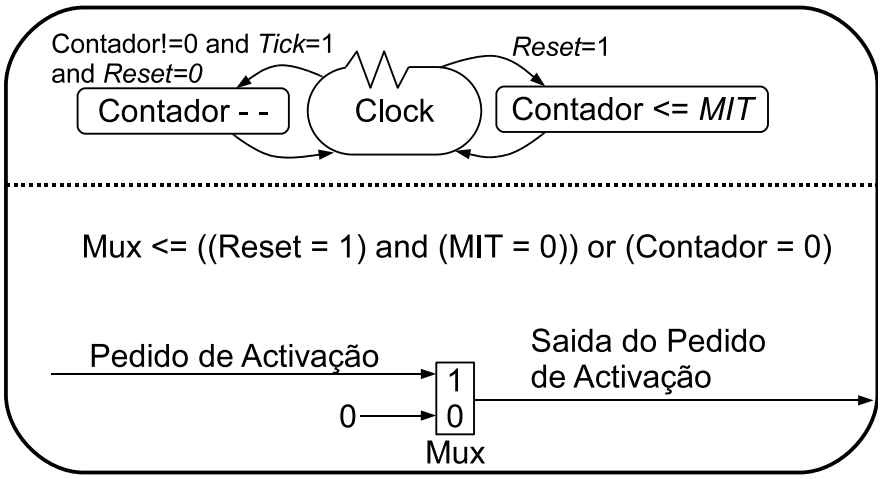


Figura 3.10: Modelo do Analisador do Intervalo Mínimo entre Activações.

3.2.4.3 Gestor do Encaminhamento de Activações

O módulo de gestão de activação de tarefas (figura 3.11), é um módulo configurável e permite fazer a interligação dos sinais de pedido de activação com as tarefas correspondentes.

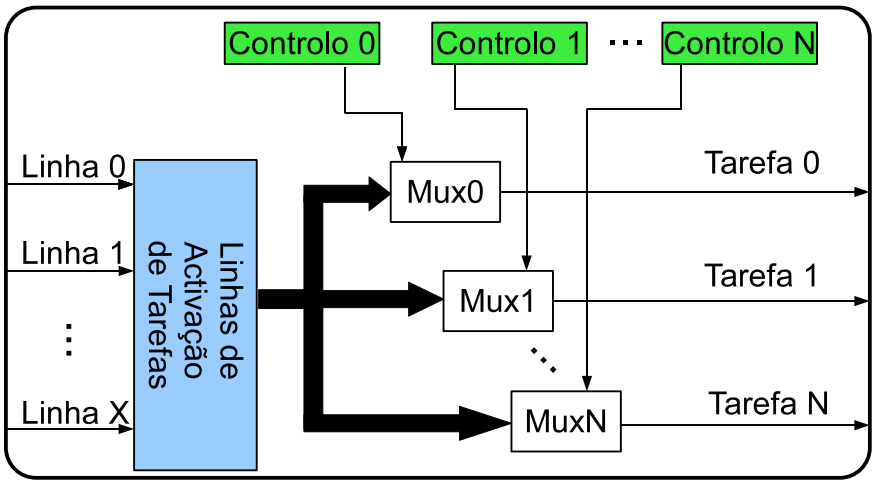


Figura 3.11: Modelo do Gestor de Encaminhamento de Activações.

O seu funcionamento, passa por agregar todas as linhas de activação de tarefas, provenientes de todos os analisadores temporais (secção 3.2.4.2), encaminhando através de *multiplexers* seleccionáveis por um sinal de controlo existente para cada linha, o pedido de activação até à tarefa correcta. Este encaminhamento é feito de forma assíncrona, por forma a não causar atrasos na propagação do sinal e funcionando estritamente como encaminhamento, tal como já foi explicado anteriormente. É de referir também, que o valor escrito nos registos de controlo é feito através da instrução *Register Task to Interruptor Line* (ver tabela 3.6), em que o valor escrito, representa qual é a linha de activação que deverá ser encaminhada para a tarefa.

No entanto, deve-se referir que uma tarefa, só pode ser activada por uma única linha de activação, no entanto, uma linha de activação pode activar múltiplas tarefas.

Ao mesmo tempo que o sinal é encaminhado para a tarefa, o gestor detecta essa mesma activação de forma paralela, e activa o sinal de *reset* do pedido de activação, no conector de interrupção e no analisador do intervalo mínimo, da linha de activação de tarefa correspondente.

3.2.5 Gestor de Tarefas

Nesta secção, é descrito como o que pode ser considerado como o núcleo do co-processador e consequentemente, o que também consome mais recursos ao nível electrónico.

O gestor de tarefas (figura 3.12) contém a tabela de tarefas, uma unidade de controlo de activação por cada tarefa, um escalonador de tarefas e ainda um *dispatcher*. Toda a arquitectura deste gestor, foi pensada com o intuito de ser paralelizada sempre que fosse possível.

Para adicionar uma tarefa, os seus atributos são colocados à entrada da tabela de tarefas, indicando qual a posição na tabela em que irá ser escrita a tarefa, e activando o bit de escrita, a tarefa é escrita na tabela.

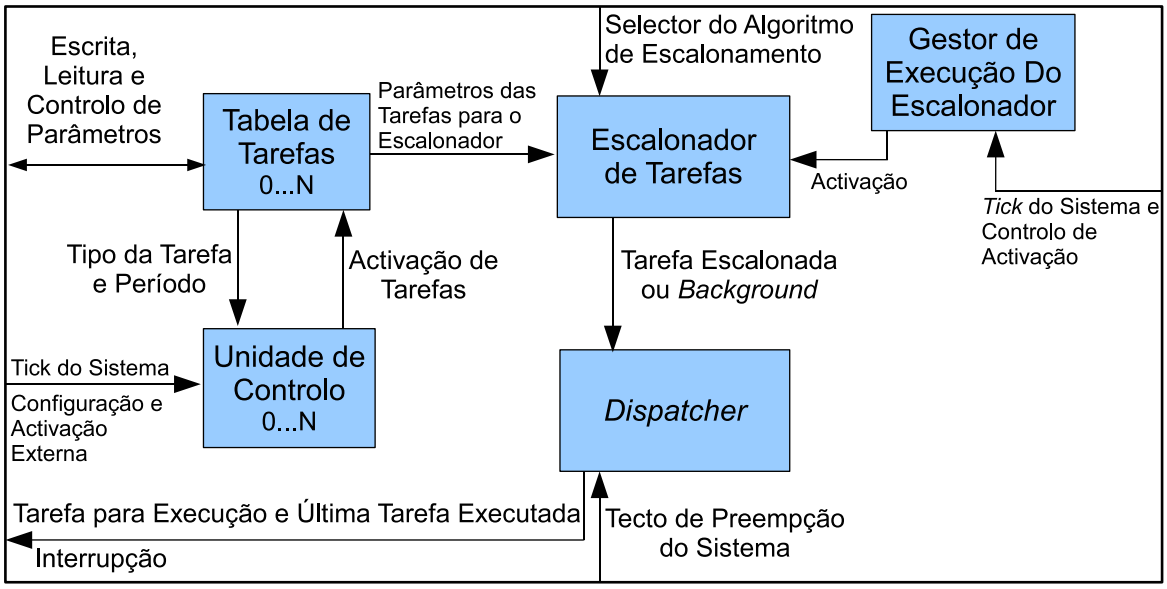


Figura 3.12: Modelo do Gestor de Tarefas.

3.2.5.1 Tabela de Tarefas

A figura 3.13 representa o modelo da tabela de tarefas.

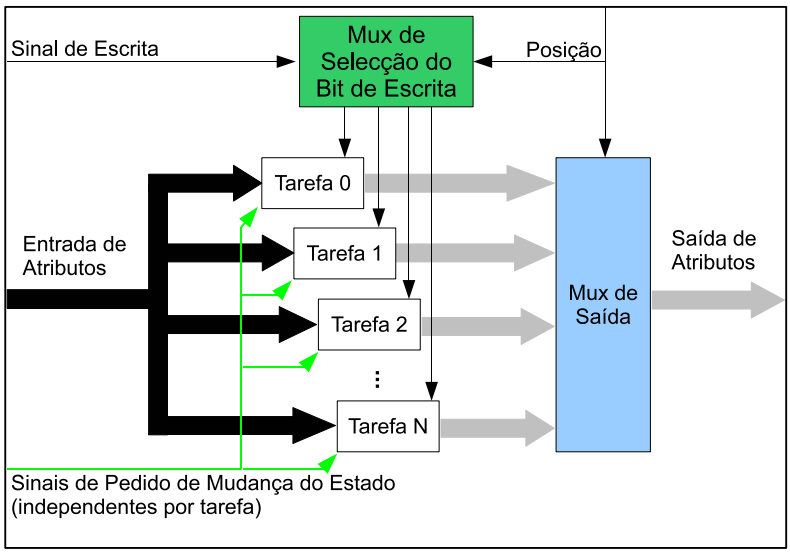


Figura 3.13: Modelo da Tabela de Tarefas.

A tabela de tarefas, contém um registo de selecção indicado no modelo com o nome *Posição*, que serve para indicar qual é a tarefa sobre a qual se irá actuar. Ao ser indicada uma posição, o *multiplexer* de saída liga os atributos da tarefa seleccionada, ao sinal de saída da tabela, enviando desta forma a informação para o descodificador de tarefas.

O sinal de posição, selecciona também a tarefa de destino do sinal de activação da escrita, que permite introduzir novos atributos e assim, adicionar uma nova tarefa ou apagar uma

tarefa existente.

Tarefas

A tabela de tarefas, implementa os atributos para cada uma das tarefas em forma de células, possibilitando o seu acesso em paralelo. Isto permite que as transições das tarefas entre estados possam ser feitas em simultâneo, sendo assim possível que várias tarefas que tenham uma activação no mesmo instante de tempo, possam transitar de estado ao mesmo tempo, evitando atrasos desnecessários de activação e execução.

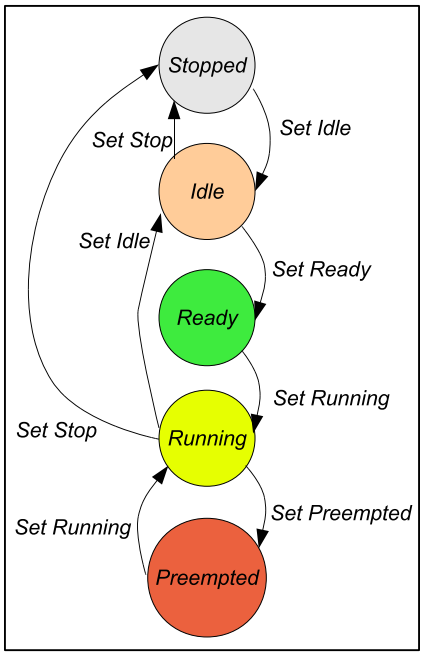


Figura 3.14: Diagrama de Estados de uma Tarefa

Cada tarefa implementa os seguintes atributos:

- Prioridade base (tarefas não periódicas) ou período(tarefas periódicas).
- Nível de preempção (tarefas não periódicas) ou *deadline* relativo (tarefas periódicas).
- *Deadline* - Instante de tempo em que a tarefa deverá ter a sua execução, concluída.
- Fase - Atraso inicial de activação da tarefa.
- *Stop Pendent* - Aviso de que a tarefa irá transitar para o estado *Stopped*, quando terminar a execução.
- *Deadline Miss* - Aviso de que a tarefa perdeu o seu *deadline*.
- *SingleShot* - Indicação que a tarefa só deverá executar uma vez, transitando depois para o estado *Stopped*.
- Tipo de Tarefa (Periódica, Aperiódica, Ordinária e Vazia (não existe tarefa presente)).
- Estado da Tarefa (ver figura 3.14).

A figura 3.14, representa o diagrama de estados de uma tarefa. A transição entre estados está implementada de forma segura, só sendo permitido a uma tarefa, transitar entre estados, quando um sinal que permite a transição, seja activado. Um pedido de transição que seja efectuado, que não esteja previsto no diagrama de estados, é ignorado não causando qualquer efeito. Por exemplo, se uma tarefa que estiver no estado *Running* e for efectuado um pedido de passagem para o estado *Ready*, o pedido será ignorado, não tendo qualquer efeito sobre o estado tarefa. Também, transições só são efectuadas se estiver presente uma tarefa(tipo da tarefa seja diferente de *Vazio*), caso contrário, os pedidos são ignorados.

A detecção de perda de *deadline*, é feita na transição do estado *Ready* para *Stopped*, em que é comparado o actual tempo presente de execução, fornecido pelo temporizador do sistema, com o *deadline* absoluto da tarefa. Caso o tempo presente seja superior, ocorreu uma perda de *deadline* e é indicado colocando o sinal *Deadline Miss* a 1, caso contrário o sinal é colocado a 0, sinalizando que a tarefa terminou em tempo útil.

Cada tarefa tem os seguintes sinais para transição entre estados:

- *SetStop* - Pede à tarefa para transitar para o estado de *Stopped*
- *SetIdle* - Pede à tarefa para transitar para o estado de *SetIdle*

- *SetReady* - Pede à tarefa para transitar para o estado de *SetReady*
- *SetRunning* - Pede à tarefa para transitar para o estado de *SetRunning*
- *SetPreempted* - Pede à tarefa para transitar para o estado de *SetPreempted*

3.2.5.2 Unidade de Controlo

A unidade de controlo (figura 3.15), é responsável pela activação da tarefa à qual está ligada. Esta unidade é constituída por um verificador/agendador temporal, e ainda, um *multiplexer* que selecciona qual a linha de activação que deve activar a tarefa, dependendo do tipo de tarefa (Periódica→Unidade de Controlo, Aperiódica→Gestor de Activações Externas, Ordinárias→Activação por *software* (descodificador de instruções)).

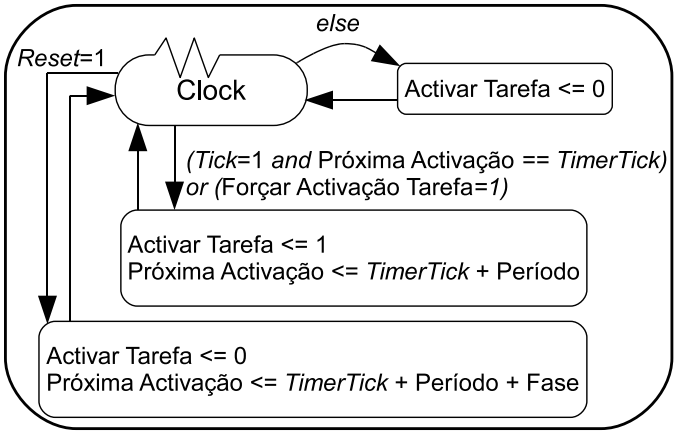


Figura 3.15: Modelo da Unidade de Controlo.

O verificador/agendador temporal, não é mais do que um comparador que compara sinal do número de *ticks* do sistema, com o sinal que contém o valor temporal correspondente à próxima activação periódica. O primeiro agendamento para execução da tarefa, determina quando é que deverá ser feita a próxima activação, usando para isso o valor actual do temporizador de *ticks* do sistema somado com o período e a fase da tarefa associada. Sendo assim, quando o comparador determina que os valores dos sinais coincidem, o sinal de activação periódica da tarefa é colocado a '1' durante um ciclo de relógio, indicando que a tarefa deverá ser activada, e ainda em paralelo, é efectuado o agendamento da próxima activação periódica, somando apenas o valor actual do temporizador de *ticks* do sistema com o período.

3.2.5.3 Escalonador

O escalonador da figura 3.16), representa o algoritmo de escalonamento paralelo, utilizado para determina a tarefa com a prioridade mais elevada num determinado momento do tempo, num sistema que suporta 8 tarefas.

A execução do algoritmo é extremamente simples. No primeiro ciclo, como a activação do escalonamento e das tarefas ocorrem em paralelo, o escalonador é obrigado a esperar pelas tarefas que foram activadas durante o seu *tick* periódico, alterem o seu estado para *ready*.

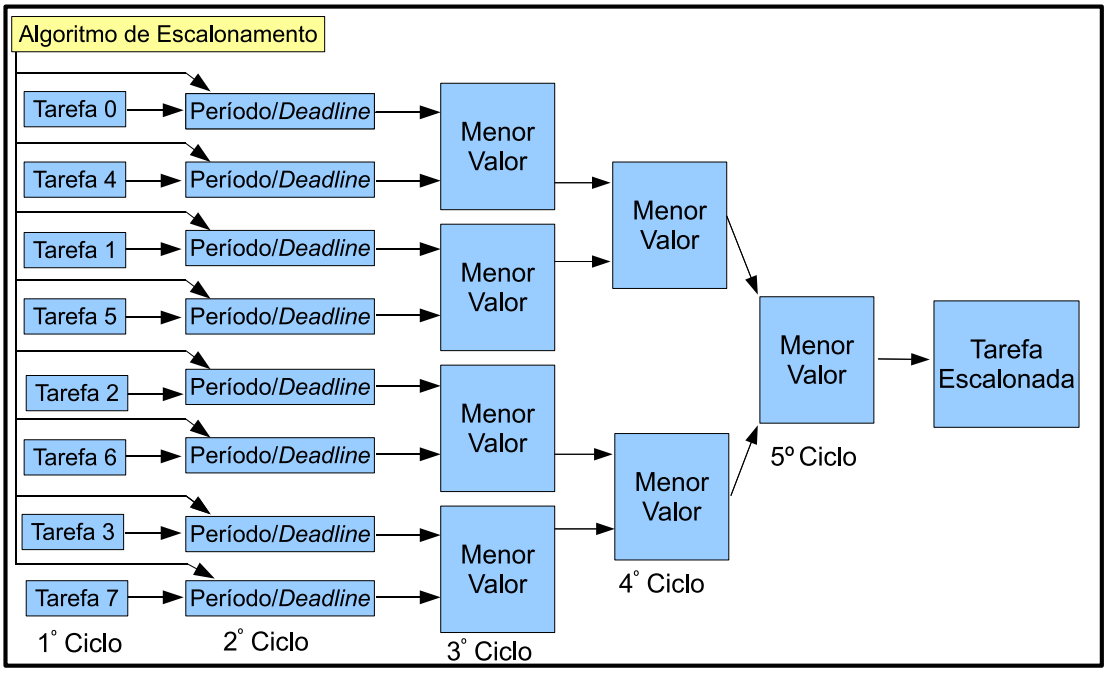


Figura 3.16: Diagrama do escalonador aplicado para 8 tarefas

Depois, para cada tarefa no sistema que esteja nos estados *Ready, Preempted e Executing*, e de acordo com a política de escalonamento (*Rate Monotonic, Deadline Monotonic ou Earliest Deadline First*), durante o segundo ciclo o escalonador selecciona o atributo temporal de cada tarefa (RM→Período, DM→*Deadline* Relativo e EDF→*Deadline* Absoluto). Se não existir nenhuma tarefa registrada ou não esteja nos estados de execução permitidos, a entrada do bloco de comparação de atributos é desactivada.

Durante o terceiro ciclo, o atributo seleccionado de cada tarefa é comparado 2 a 2, reduzindo o número de atributos a comparar no próximo ciclo em metade. No quarto ciclo, os atributos de cada tarefa que vêm do ciclo anterior, são novamente comparados 2 a 2, reduzindo para duas tarefas a comparar durante o quinto e último ciclo, em que é feita a última comparação, sendo obtida a tarefa com maior prioridade.

Caso durante a execução do algoritmo, um dos blocos de comparação(bloco menor valor) não contenha atributos presentes em nenhuma das entradas, a saída do respectivo bloco é desactivada, indicando ao comparador seguinte que não foi obtido um resultado. Isto significa que no último ciclo de execução, o algoritmo não obtenha uma tarefa, significando que na realidade, não existe nenhuma tarefa para ser executada e que é o *background* que deverá ser colocado no contexto do processador.

A complexidade do algoritmo é $O(logN)$ e utilizando a formula matemática

$$Ciclos = 2 + log_2 N$$

podemos determinar o número exacto de ciclos que o algoritmo requer para obter um resultado, em ordem ao número máximo de tarefas que o co-processador suporta (N). Assim garante-se, que para um determinado número de tarefas, o algoritmo de escalonamento irá levar sempre o mesmo número de ciclos para obter um resultado, tornando-se assim, verdadeiramente determinístico.

3.2.5.4 *Dispatcher*

O *dispatcher*, é o módulo que efectivamente toma a decisão, de interromper ou não, o processador para uma possível mudança de contexto. O *dispatcher* passa primeiro por verificar se é uma tarefa ou se é o *background* que irá ser colocado no contexto do processador, verificando se a saída do escalonador, está activa ou não.

Caso não esteja activa, significa que não existe uma tarefa pronta para execução e deverá ser o *background* a ser colocado no contexto do processador. Caso esteja activa, então o escalonador estará a indicar a tarefa com maior prioridade, nesse dado momento de execução do sistema. No entanto, antes de o processador ser interrompido, é feita a verificação de se a tarefa que está actualmente presente no contexto do processador, é igual à tarefa obtida pelo escalonador. Se for, significa que a actual tarefa em execução é a de maior prioridade e assim, não é feita qualquer interrupção do processador, deixando que a tarefa continue a sua normal execução, caso contrário, significa que existe uma nova tarefa com prioridade superior, procedendo-se à interrupção do processador para que seja feita a troca de contextos.

Caso especial de execução

O *dispatcher* está sujeito a um caso especial quanto ao lançamento da interrupção do processador. Quando uma tarefa termina, a instrução de terminação da tarefa (secção 3.2.2.1), é feito um pedido para execução do escalonador e consequentemente, do *dispatcher*, assim, como não faz parte de uma execução periódica do escalonamento, a interrupção gerada é inibida, devido ao simples facto de a mudança de contexto já ser contemplada durante a execução da componente de *software* da instrução.

3.2.5.5 Gestor de Execução do Escalonador

O gestor do *tick* do sistema, encarrega-se de gerir os *ticks* criados pelo temporizador do sistema, enviando-os ao escalonador. Este gestor é responsável por inibir ou adiar a activação do escalonador(ver secção 3.2.5.3), que através do *dispatcher* (ver secção 3.2.5.4) provoca a interrupção do processador.

A inibição do escalonamento no ponto de vista do co-processador, pode ser comparada como uma inibição de interrupções ou de preempção quando vista do ponto do executivo, visto que deixa de ser executado o escalonador e consequentemente, deixa o processador de ser interrompido.

Existe no entanto, o ganho de um beneficio devido à capacidade de paralelismo do co-processador. A inibição de interrupções do co-processador, não afecta nem a gestão temporal das tarefas periódicas, nem a gestão do intervalo mínimo de activação das tarefas aperiódicas!

Foi referido anteriormente, que a execução do escalonador pode ser adiada. Isto acontece, devido à necessidade de garantir que o escalonador não é activado ou re-activado, durante a execução ou de uma instrução ou do próprio escalonador (caso que pode acontecer durante activações consecutivas de tarefas periódicas e aperiódicas). Sendo assim, o gestor durante a execução do escalonador, guarda um possível pedido de execução, que é passado ao escalonador quando este terminar a sua execução.

A arquitectura desde gestor, como se pode ver pela figura 3.17, é bastante simples. O gestor contém um selector que encaminha para a saída, um sinal 0, um sinal 1 e um sinal

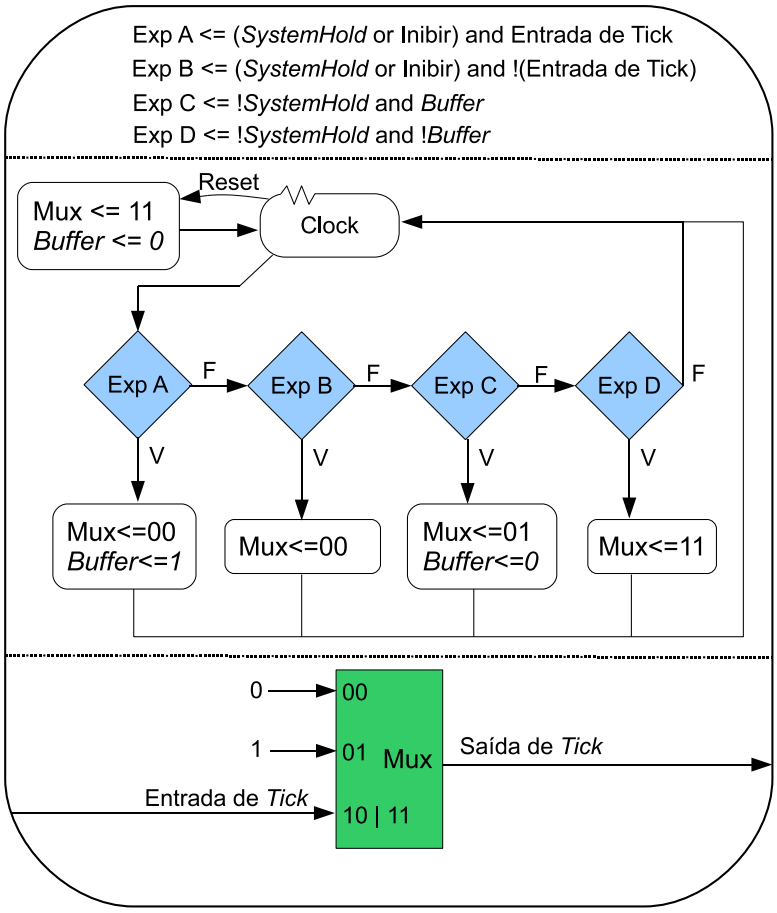


Figura 3.17: Diagrama do funcionamento do Gestor de Activação do Escalonador.

que corresponde ao *tick* proveniente do gestor temporal, contendo também, um circuito combinatório sincronizado pelo *clock* global responsável por seleccionar qual a entrada do selector que é encaminhada para a saída.

Quando é necessário inibir a preempção de tarefas, o sinal *Inib* é colocado a 1, indicando que existe uma suspensão da execução do escalonador. Neste momento, qualquer pedido de activação que apareça, será guardado no *Buffer*. O sinal *System Hold* tem um efeito semelhante. Existe devido ao facto das primitivas de utilização do co-processador, serem atómicas. Quando este sinal está activo, significa que está presentemente a ser executada uma primitiva e sendo assim, o escalonador deverá ser inibido de executar.

No momento em que o sinal *System Hold* fica igual a 0, é verificado se o *buffer* contém um pedido de activação, e caso contenha, o sinal de saída de *tick* é colocado a 1 durante um ciclo de relógio, enviado assim um pedido de execução ao escalonador.

3.2.6 Controlador de Semáforos

Para sincronização de recursos, o co-processador implementa um controlador de semáforos, que permite que uma tarefa se registre com o seu nível de prioridade. A partir daí, a tarefa poderá bloquear e libertar o semáforo, subindo e descendo respectivamente, o tecto do

sistema.

Este controlador, utiliza um modelo simplificado da Política de Pilha de Recursos (secção 2.1.3.2), por forma a reduzir a quantidade de recursos que seriam necessários, quando comparado com uma implementação completa. Sendo assim, existem algumas restrições quanto ao seu uso, em que a ordem de libertação dos semáforos deve ser inversa em relação à ordem de fecho (*Bloquear A - Bloquear B - Libertar B - Libertar A*). Também não é possível remover o registo de uma tarefa especifica de um semáforo, sendo apenas possível reiniciar o semáforo.

A implementação de um controlador de semáforos, permite que a gestão dos recursos cujo acesso deverá ser mutuamente exclusivo, seja feita ao nível do hardware, interligando com o *dispatcher* (ver secção 3.2.5.4) o tecto de preempção do sistema.

A figura 3.18, representa a arquitectura do controlador de semáforos. O controlador é constituído por um gestor e uma pilha (*stack*), cujo valor presente no topo, corresponde ao tecto de preempção do sistema.

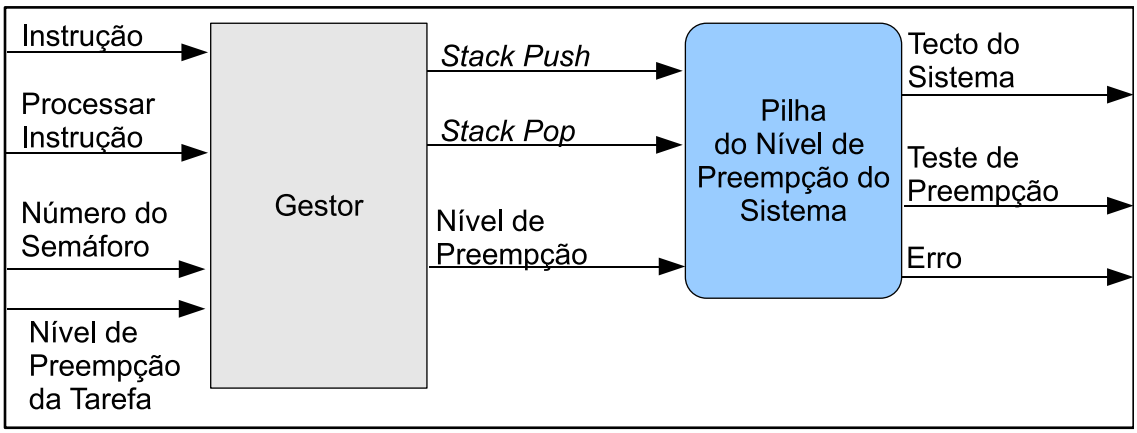


Figura 3.18: Modelo do controlador de semáforos.

Neste modelo, o tecto de preempção existe perante um modelo de prioridade invertida, pois genericamente, o nível de preempção deveria ser directamente proporcional, correspondendo o menor valor à menor prioridade e o maior valor, à maior prioridade. No entanto, com o intuito de simplificar a arquitectura e devido ao facto de o nível prioridade de uma tarefa, ser inversamente proporcional em relação ao seu atributo temporal seleccionado pelo algoritmo de escalonamento (RM→Período, DM→*Deadline* Relativo e EDF→*Deadline* Absoluto), o tecto inicial de preempção de cada semáforo é o valor máximo representável numa precisão de 32 bits, para valores inteiros sem sinal. Sendo assim, o tecto de preempção inicial de cada semáforo e o tecto inicial do sistema na precisão indicada, é igual a 0xFFFFFFFF.

Observe-se agora a figura 3.19. Para seleccionar a acção a ser executada pelo controlador de semáforos, é usado um sinal de 2 bits que permite codificar 4 acções (ver também a secção 3.2.2.1):

- Registo de uma Tarefa num Semáforo (*Register Semaphore*).
  - Requer como argumento, o nível de preempção da tarefa.
- Fecho de um Semáforo (*Lock Semaphore*).
  - Requer como argumento, o número do semáforo a fechar.

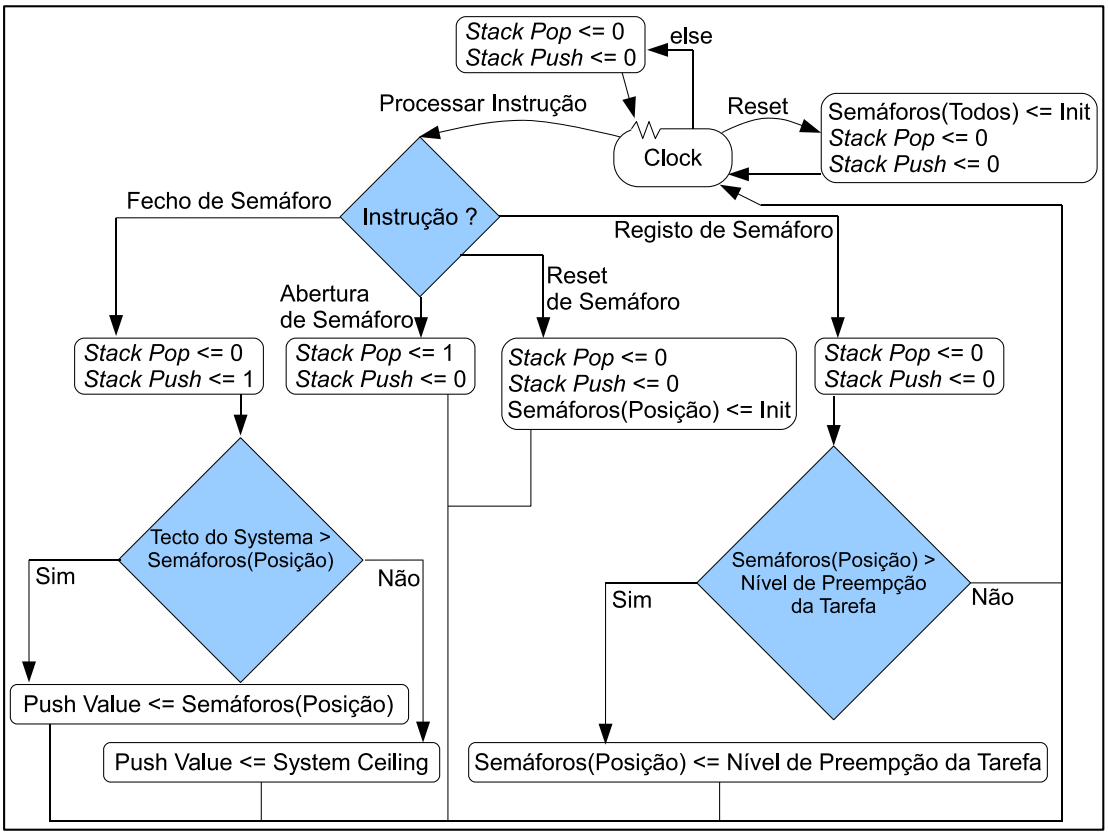


Figura 3.19: Diagrama do funcionamento do gestor do controlador de semáforos.

- Abertura de um Semáforo (*Unlock Semaphore*).
- Reinicialização do Nível de Preempção de um Semáforo (*Reset Semaphore*).
  - Requer como argumento, o número do semáforo a reiniciar.

Para executar uma dada acção pretendida, o descodificador de instruções do co-processador (ver secção 3.2.2), durante um ciclo de relógio, atribui ao sinal de instrução do controlador, o valor da codificação correspondente à acção, indicando ao que o controlador deve executar a acção indicada, colocando o sinal Processar Instrução a 1.

Quando um semáforo é libertado, é dada ordem à pilha para remover o nível de preempção que se encontra no topo. Isto provoca a activação do sinal de Teste de Preempção. Este sinal está ligado ao escalonador de tarefas, e serve para indicar que deve ser feito um novo escalonamento, para verificar se a tarefa que continua em actual execução, continua a ter manter a prioridade mais elevada.

A pilha contém um número limite de posições, sendo que em caso de abusos por parte das tarefas, é possível atingir o tamanho máximo, ou inversamente, ficar vazia. Para protecção, a pilha tem um sinal de saída usado para sinalizar a ocorrência de um erro, durante a fase de fecho e abertura de um semáforo. Quando é feito um fecho de semáforo e o sinal de erro é activado, significa que a pilha atingiu o seu limite. Se o sinal de erro for activado durante a abertura de um semáforo, significa que a pilha está vazia.

É importante referir, que como todos os semáforos estão implementados em *hardware*, sendo-lhes conferido um nível de preempção igual ao nível mais baixo do sistema no momento do arranque do sistema, e como o nível de preempção de um semáforo só influência o tecto do sistema no momento em que é bloqueado, não faria sentido indicar se um semáforo estaria activo ou não, quando o significado de activo corresponde a estar alocado ou não, como acontece na implementação do executivo OReK em *software* (ver anexo A - secçãoA.4.2.6). Sendo assim, decidiu-se assim simplificar o módulo de semáforos e não existir distinção entre semáforos activos e semáforos inactivos.

Capítulo 4

Avaliação do Desempenho

Neste capítulo irá ser apresentada a avaliação temporal do co-processador, com a respectiva análise dos resultados obtidos. Inicialmente é feita uma descrição completa da configuração do sistema com o co-processador que é avaliado, sendo também feita uma breve descrição do sistema que utiliza o mesmo executivo mas sem o co-processador.

De seguida, serão apresentadas várias tabelas contendo os resultados da avaliação temporal das primitivas do executivo. É também feita a comparação dos resultados obtidos, com os resultados do sistema sem co-processador, onde será feita uma análise posterior explicando os valores obtidos. No final é também apresentada uma tabela onde mostra os recursos consumidos, por cada uma das parametrizações do co-processador usadas.

O sistema a analisar irá consistir no executivo OReK modificado para suportar o co-processador implementado na FPGA da plataforma XUPV2Pro. Os resultados desta análise serão comparados com os resultados da análise de funcionamento do executivo OReK sem o auxilio do co-processador, que já foi efectuada em [dS08].

Todos os resultados relativamente à análise do executivo OReK sem a utilização do co-processador, são da autoria de [dS08] e que já foram apresentados e defendidos perante um júri, tendo sido posteriormente considerados válidos. Os dados obtidos pelo autor, são utilizados para efeito de comparação com os resultados do sistema que irá ser avaliado, com o objectivo, de determinar efectivamente o quão útil poderá ser a utilização do co-processador, no melhoramento do desempenho e determinismo do executivo.

4.1 Introdução

Como já foi explicado no capítulo 2.5, os sistemas de tempo-real diferenciam-se dos sistemas genéricos não só pela sua precisão temporal na gestão das suas tarefas, como também pelo determinismo das suas acções. Para que se possa determinar o desempenho e determinismo de um executivo de tempo-real, é necessário fazer primeiro uma análise do tempo de execução das suas primitivas básicas de execução. A análise dos resultados, permite conhecer as possíveis fontes de indeterminismo que possam existir e também, conhecer o desempenho das próprias primitivas do executivo de tempo-real.

O Executivo OReK

O OReK, cuja arquitectura está representada na figura 4.1, é um executivo de tempo-real completamente preemptivo, que implementa uma camada de abstracção, e permite fornecer serviços de temporização, gestão de tarefas e sincronização no acesso a recursos partilhados.

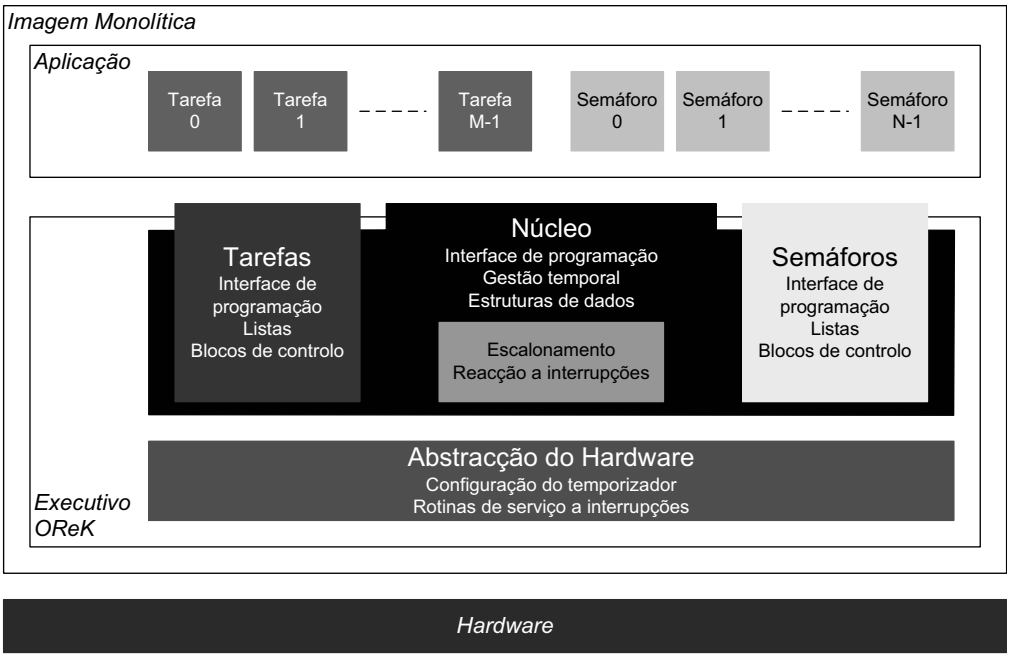


Figura 4.1: Arquitectura do Executivo OReK - Implementação exclusiva em *software* (OReK-PPC-Int e OReK-PPC-*Cached*).

Os serviços disponibilizados pelo executivo, dividem-se nos seguintes quatro grupos:

- Configuração e estado (arranque, terminação e informação temporal).
- Diagnóstico (códigos de erro e mensagens).
- Gestão de tarefas (criação, destruição, manipulação, etc).
- Gestão de semáforos (criação, destruição, manipulação, etc).

Para a implementação de tarefas, este executivo contempla tarefas periódicas de tempo-real (críticas e não críticas), aperiódicas de tempo-real (críticas e não críticas) e ainda, tarefas ordinárias, sendo possível utilizar para o escalonamento das mesmas, as políticas *Rate Monotonic* (RM), *Deadline Monotonic* (DM), *Earliest Deadline First* (EDF) e *First Come-First Served* (FIFO).

Para garantir o acesso exclusivo a recursos partilhados por parte das tarefas, o executivo OReK implementa semáforos binários, sendo estes geridos pela política de pilha de recursos (*SPR*).

A camada de abstracção fornecida pelo executivo, permite a utilização dos seus serviços através de primitivas de acesso, que se encontram implementadas em *software* para as implementações OReK-PPC-Int e OReK-PPC-*Cached*, estando implementadas em *hardware* na implementação OReK-CP.

A avaliação irá centrar-se nas primitivas de acesso aos serviços do executivo. Isto contempla todas as primitivas relacionadas com o arranque e configuração do executivo, gestão e

manipulação de tarefas e semáforos, sendo ainda contempladas as primitivas de configuração do gestor de atendimento de pedidos de interrupção externos e rotinas de mudança do contexto e terminação de tarefas.

4.2 Configuração do Sistema

Nesta secção é feita a apresentação do sistema que irá ser testado. Detalhes sobre a sua estrutura e configuração, são aqui apresentados e explicados.

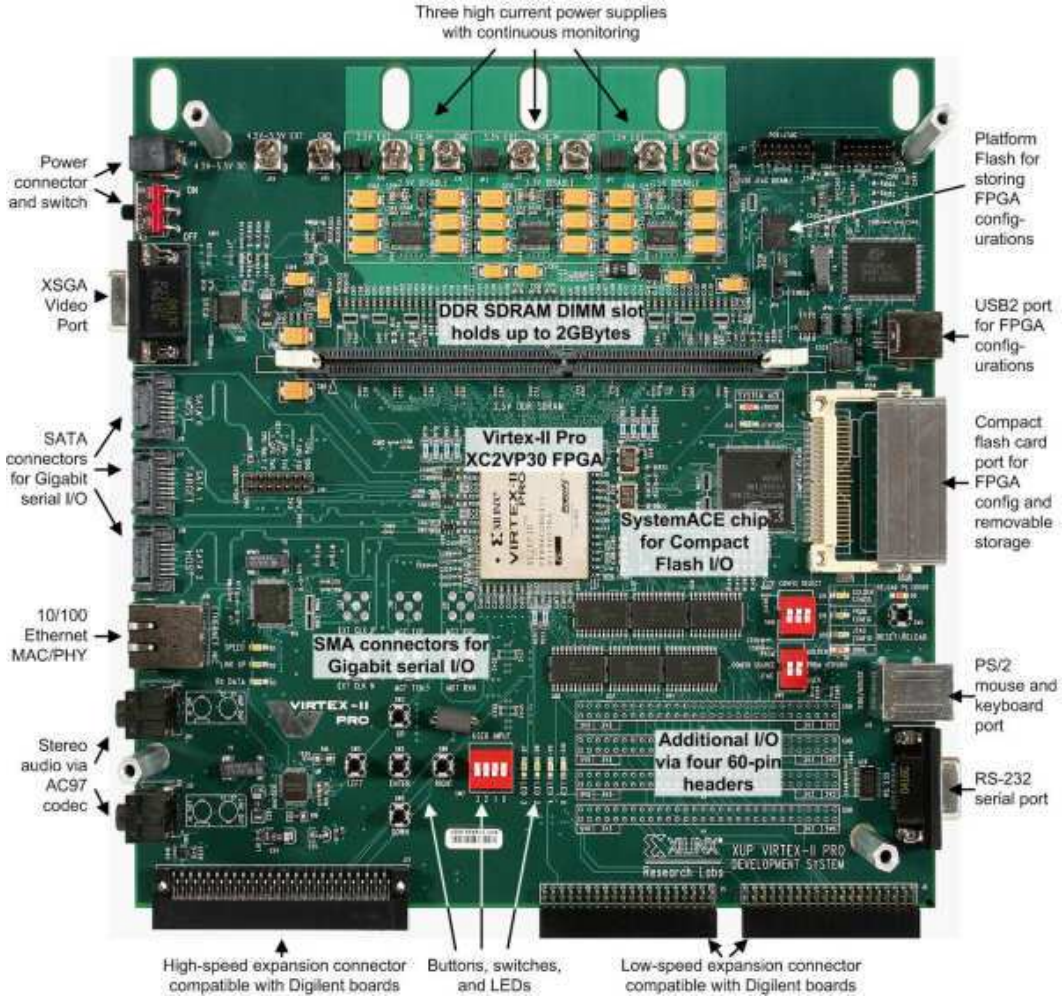


Figura 4.2: FPGA Virtex 2 Pro (XUPV2Pro)[Dig08].

4.2.1 A plataforma

Para avaliar e validar a utilização do co-processador, é necessário uma plataforma reconfigurável que contenha os seguintes atributos:

- Uma FPGA.
- Presença de um Processador.
- Memória Principal.
- Suporte de dispositivos de entrada e saída de informação (botões de pressão e escrita na porta série, como exemplo).

Para este trabalho, foi disponibilizada a plataforma Virtex 2 Pro do fabricante Digilent (figura 4.2), também usada no trabalho apresentado [dS08]. Esta plataforma de desenvolvimento, contém as seguintes características:

- Uma FPGA XC2VP30 que contém 30816 Células Lógicas, 136 multiplicadores de 18-bit, 2448Kb em blocos de RAM e ainda um processador PowerPC 405.
- Um *slot* para memória DDR SDRAM com suporte até 2Gb.
- Uma ligação Ethernet 10/100 Mb/s.
- Uma ligação USB 2.0.
- Um *slot* para ligação a um cartão de memória *flash*.
- Uma saída de video XSGA e uma saída de Audio.
- Portos SATA, PS/2 e RS-232.
- Portos de expansão de baixa e alta velocidade para conexão a periféricos exteriores.
- 4 Botões de Pressão, 5 Interruptores e 4 Led's.

4.2.2 Processador PowerPC 405

O processador usado foi implementado de forma *hardwired* na FPGA XC2VP30. Escolheu-se este processador, devido ao facto da sua utilização minimizar os recursos consumidos e ainda, devido à sua frequência de execução ser superior. O PowerPC 405 é um processador do tipo RISC que contém os seguintes atributos:

- Suporte de Instruções de 32bits, sendo a sua execução feita num *pipeline* de 5 estágios.
- Possibilidade de endereçar até 2GB de memória externa.
- Memória *cache* programável.
- Um sinal para recepção de interrupções externas, programável.

Ainda em relação ao processador, o sinal de recepção de interrupções externas foi ligado directamente ao sinal de saída de interrupções geradas pelo co-processador, permitindo assim um controlo directo sobre as interrupções geradas sobre o processador.

4.2.3 Memória do Sistema

Para instalar e correr o executivo de tempo-real OReK, é necessária a presença de memória. Dentro da FPGA é possível construir uma memória, utilizando os blocos de RAM disponibilizados pela mesma e assim, construir memórias dedicadas para o segmento de instruções, segmento de dados e memória dinâmica do sistema (*heap* e *stack*).

No entanto, para o sistema de teste, são construídas duas memórias independentes sendo a primeira, uma memória utilizada pelo segmento de instruções e a segunda, uma

memória onde reside o segmento de dados, a *heap* e a *stack*. Optou-se por este método, devido a restrições impostas pela utilização da memória *cache*, que impossibilita o *caching* do segmento de dados e da memória dinâmica, caso estejam separadas, bloqueando o sistema.

Sendo assim, a estrutura e segmentação da memória é a seguinte:

- Memória para Instruções com 128KB de espaço.
- Memória Principal com 64KB de espaço:
  - 32KB para o segmento de dados.
  - 16KB para a memória *heap*.
  - 16KB para a memória *stack*.

4.2.4 Memória *Cache*

A memória cache é uma memória intermédia que o processador possui internamente, que lhe permite ter uma pequena porção da memória principal para que o acesso seja feito de forma mais rápida.

Cada vez que existe um acesso à memória principal, um bloco dessa memória é copiado para a cache do processador, acelerando os acessos futuros. No entanto, quando um determinado endereço de memória não se encontra presente na cache, é necessário ser feito uma espera para que um novo bloco da memória principal, seja copiado para a cache e assim, ser possível lêr o valor presente no endereço de memória pedido. Ao acesso a um endereço de memória que está presente na cache, dá-se o nome de *hit* e ao acesso a um endereço que não está presente, dá-se o nome de *miss*.

No entanto, a utilização da memória *cache* pode ser um problema grave nos sistemas de tempo-real. O facto de ser muito difícil prever se um determinado endereço de memória se encontra presentemente mapeado na *cache*, o que faz variar os tempos de acesso aos dados ou seja, introduz indeterminismo. Este indeterminismo provoca a degeneração da qualidade de um sistema de tempo-real.

4.2.5 Periféricos Utilizados

A plataforma Virtex 2 Pro, como já foi anteriormente explicado, disponibiliza vários tipos de periféricos, desde porta VGA, Ethernet, Memória flash, entre outros. No entanto, devido à presença de um circuito reconfigurável (FPGA), existe a possibilidade de, em vez de estarem presentes na plataforma os *chips* de controlo dedicados para cada um dos periféricos, estes irão ser implementados na FPGA através da sua descrição em linguagens apropriadas (VHDL ou Verilog).

No entanto, quantos mais periféricos forem seleccionados para utilização com o processador do sistema, mais recursos da FPGA irão ser necessários para a síntese dos seus respectivos controladores. Como a FPGA tem recursos limitados, quantos mais periféricos forem seleccionados para uso, menor serão os recursos disponibilizados para outras utilizações, sendo para este caso, a necessidade de implementar o co-processador.

Sendo assim, para efectuar os testes necessários com as tarefas aperiódicas, era necessário simular pedidos de activação de tarefas, como se estes fossem gerados por periféricos externos. A melhor opção, foi utilizar os botões de pressão e interruptores, disponibilizados pela plataforma, que poderão ser ligados directamente ao co-processador, minimizando a quantidade de recursos necessários, cumprindo assim o objectivo.

Como dispositivos de output de informação, foram utilizados os 4 *Led's* presentes na plataforma, assim como a porta série. Apesar de ser necessário a síntese de um controlador próprio para a porta série, a sua presença permite um output mais personalizado e menos limitado, como é o caso dos 4 *Led's*.

4.2.6 Barramento Local

Para interligar todos os periféricos, incluindo o co-processador que irá ser testado, é utilizado o *Processor Local Bus* versão 4.6 [Xil07]. No entanto, é preciso ter em consideração que este barramento local, para que seja feita uma gestão dos acessos de uma forma correcta, introduz uma latência considerável no acesso aos periféricos, o que afecta os resultados obtidos na utilização das primitivas do executivo que usam o co-processador. Esta latência, é tida em conta ao analisar os resultados, sendo no entanto, os seus efeitos previsíveis (ver secção 4.4.1).

A FPGA irá funcionar a uma frequência de 100Mhz, sendo inferior à frequência de funcionamento do processador. Assim, irá existir alguma latência no acesso a periféricos externos, à memória principal e à memória de dados por parte do processador. Apesar da existência de latência, esta é determinística.

4.2.7 Parametrização do Co-Processador

O co-processador permite ser especializado através de parâmetros, no acto da síntese. Isto permite adaptar o co-processador tanto ao trabalho que irá efectuar, assim como, à quantidade de recursos disponíveis no circuito reconfigurável a usar. Existem no entanto, algumas limitações nos parâmetros a indicar. De notar, que para além da limitação imposta pela própria arquitectura, existe também a limitação dos próprios recursos do circuito reconfigurável.

4.2.7.1 Número de Tarefas

O número de tarefas deverá ser potência de base 2 no intervalo de valores $[4, 256]$, isto é, só os valores $[4, 8, 16, 32, 54, 128, 256]$ é que são possíveis de ser utilizados como valores válidos.

A razão desta limitação, está na implementação do escalonador, que é uma implementação em forma de árvore binária, o que requer que o número de tarefas base seja potência de base de 2. A limitação do valor máximo, é devido ao número de bits do sinal de selecção de tarefa, que se preferiu limitar assim por forma a simplificar a arquitectura.

4.2.7.2 Número de Linhas de Interrupção Externas

O número de linhas de atendimento a interrupções externas, está limitado a um máximo de 255 linhas. A limitação é imposta pelos sinais internos de acesso ao gestor de interrupções externas, que por uma questão de simplificação da arquitectura, preferiu-se limitar a gama de valores possíveis de indicar.

É de lembrar, que para cada linha é implementado um conector de interrupções (secção 3.2.4.1) e um analisador temporal (secção 3.2.4.2).

Concluindo, a gama de valores que pode ser possível de indicar no parâmetro do co-processador, é representada pelo intervalo [0; 255].

4.2.7.3 Número de Semáforos Disponíveis

O número de semáforos disponíveis, mais uma vez está limitado a um máximo de 255 semáforos. A limitação, mais uma vez, é imposta também pelo acesso ao gestor de semáforos, também por razões de simplificação da arquitectura.

Mais uma vez, a gama de valores que pode ser possível de indicar no parâmetro do co-processador, é representada pelo intervalo [0; 255].

4.2.7.4 Configurações Usadas na Avaliação

Como a arquitectura do co-processador foi estruturada de forma a maximizar o paralelismo, existem certos parâmetros que não causam alterações no seu funcionamento. Estes parâmetros são, o número de linhas de interrupção externas e também, o número de semáforos.

Sendo assim, para todos os testes, o número de linhas de interrupção externas será estático (NUMBER_INTERRUPTIONS=5) sendo também, o número de semáforos estático (NUM_SEMAPHORES=4).

No entanto, o número de tarefas suportadas pelo co-processador, faz variar o escalonamento de tarefas como já foi explicado na secção 3.2.5.3. Para as primitivas que dependem da utilização do escalonador de tarefas, são feitas três análises, correspondendo a três parametrizações diferentes suportando 4, 8 e 16 tarefas respectivamente (NUM_TASKS=4, NUM_TASKS=8, NUM_TASKS=16). Não foi possível fazer análises para um número de tarefas superior devido à escassez de recursos disponibilizados pela FPGA.

4.3 Sistema de Comparação

Para que se possa fazer uma comparação de resultados, com a finalidade de saber se efectivamente existe um melhoramento do desempenho e determinismo, é necessário ter um sistema base cujos resultados possam ser comparados de forma justa.

Em [dS08], o autor faz a análise do funcionamento do executivo OReK, sem qualquer auxilio de um co-processador, utilizando vários processadores diferentes (ARPA-MT [ASRdO07], MicroBlaze [Xil08a] e PowerPC 405 [Xil08c]). Desse conjunto de processadores, só serão utilizados os resultados de uma pertencendo ao processador PowerPC 405, e cuja configuração se encontra em duas versões distintas, uma utilizando a memória *cache* do processador (OReK-PPC-*Cached*), sendo a outra uma versão mas que não usa a memória *cache* (OReK-PPC-Int).

4.3.1 Configuração da Plataforma

O sistema apresenta a seguinte configuração:

- Utilização do processador PowerPC 405 da plataforma XUPV2Pro.
- Memória interna construída através de *BlockRams* disponíveis na FPGA, ligadas ao processador através de uma ligação dedicada.
- Utilização de um Temporizador programável sintetizado, para geração de interrupções periódicas.

- Frequência de operação do processador igual a 300 MHz.
- Frequência de operação da FPGA igual a 100 MHz.

4.4 Análise Temporal

Para determinar o tempo de execução de cada primitiva do executivo, é necessário utilizar um temporizador que disponibiliza-se uma elevada precisão temporal. Sendo assim, foi utilizado o temporizador interno do CPU, que irá funcionar à mesma frequência do núcleo, sendo de 300MHz no caso do processador PowerPC 405, o que significa que as leituras dos registos temporais irão ter uma precisão de 3,3ns. Isto permite obter resultados temporais precisos

É feita a leitura do registo temporal antes da execução da primitiva, sendo também feita uma leitura após a finalização da primitiva, em que depois é subtraído à leitura da finalização, a leitura obtida antes da execução, determinando-se assim o tempo de execução da primitiva, com uma precisão igual ao período do relógio à qual funciona o processador.

As leituras dos valores são feitas utilizando código *assembly*, acedendo directamente aos registos temporais do processador, sendo o valor da primeira leitura, armazenado no segmento de dados.

Em relação ao teste das primitivas, é necessário ter em conta que devido à presença de um co-processador, grande parte das acções internas do executivo, são feitas em *hardware*. Isto significa uma redução drástica do tempo de execução de código, onde se torna efectivamente mais visível na mudança de contexto, terminação de uma tarefa mas especialmente, no escalonamento de tarefas, cujas acção está completamente implementada dentro do co-processador.

Deve ser ainda referido que em [dS08], o autor apresentou os resultados da avaliação temporal por acção (ex. salvaguarda de contexto, selecção da próxima tarefa, entre outras), ao invés de ser por execução de primitiva (ex. mudança de contexto da tarefa, terminação da tarefa) como está apresentada a avaliação do co-processador, neste capítulo.

Isto significa que para ser feita uma comparação justa, irá ser necessário especificar todos os tempos das acções que constituem a execução de uma primitiva. Assim, as tabelas relativamente aos sistemas OReK-PPC-Int e OReK-PPC-*Cached* apresentam os resultados temporais para cada acção, estando isoladamente no final da tabela, o valor final correspondente à execução da primitiva em análise.

Nas próximas secções, serão descritas quais as acções que uma dada primitiva efectua e qual o seu impacto temporal.

4.4.1 Tempos de Comunicação

Antes de se iniciar a avaliação das primitivas, é necessário saber primeiro qual a latência inserida no tempo de execução das primitivas, cada vez que se acede a um recurso externo. Sendo assim, este teste irá mostrar, quanto tempo demora a lêr e escrever na memória principal e nos registos internos do co-processador.

Na tabela 4.1, encontram-se os resultados dos testes de escrita e leitura na memória e no co-processador. Isto permite saber qual a latência gerada, na execução das primitivas.

Acção Avaliada	Implementação OReK-CP	
	Sem *Cache*	Com *Cache*
Leitura da Memória (uma palavra de 32 bits)	$0,1221\mu s$	$0,0099\mu s$
Escrita na Memória (uma palavra de 32 bits)	$0,0198\mu s$	$0,0198\mu s$
Leitura do Co-Processador (um registo de 32 bits)	$0,1386\mu s$	
Escrita no Co-Processador (um registo de 32 bits)	$0,6642\mu s$	

Tabela 4.1: Acessos aos recursos do sistema.

Observando os resultados, verifica-se que tanto a leitura como a escrita de um registo no co-processador têm uma elevada latência, o que influência fortemente todas as primitivas que necessitem de aceder ao co-processador. Os resultados no acesso à memória, também demonstram alguma latência de acesso, no entanto, não são tão críticos como os acessos ao co-processador.

No entanto, a latência no acesso ao co-processador pode ser efectivamente reduzido, caso uma ligação dedicada entre o processador e o co-processador exista. Isto iria permitir obter tempos de interacção com o co-processador muito inferiores aos actuais, que utilizam o barramento local para comunicação, melhorando assim os tempos de desempenho das acções que interagem com o co-processador. Os problemas da latência podem ser observados na avaliação das primitivas do executivo nas secções 4.4.6 e 4.4.7.

4.4.2 Temporização da Mudança de Contexto

A primitiva de mudança de contexto é utilizada para efectuar a troca de contexto entre duas tarefas de aplicação ou entre o *background* do sistema e uma tarefa de aplicação, caso seja o *background* que esteja em actual execução.

Esta primitiva é executada cada vez que o co-processador gera uma interrupção do processador, devido a ter determinado que existe uma tarefa para execução, que tem uma prioridade superior à tarefa que se encontra presente no contexto do processador.

É de referir, que no momento em que a interrupção é gerada pelo co-processador, já terá sido efectuado o escalonamento de tarefas, já estando presente no registo apropriado (ver secção 3.2.1), a referência de qual a próxima tarefa a executar.

As acções da primitiva de mudança de contexto são:

- Salvaguarda do contexto da actual tarefa em execução.
- Leitura da referência da próxima tarefa a executar, do registo apropriado pertencente ao co-processador.
- Carregamento do contexto da nova tarefa, para o processador.

Parâmetro Avaliado		Implementação OReK-CP		**# Tarefas**
		Sem *Cache*	Com *Cache*	
Mudança de Contexto da Tarefa	BG→Tarefa	$7,870\mu s$	$1,323\mu s$	Qualquer
	Tarefa→Tarefa	$7,893\mu s$	$1,343\mu s$	

Tabela 4.2: Sistema Com Co-Processador - Mudança de Contexto da Tarefa

Parâmetro Avaliado	Implementação OReK (*Software*)		# Tarefas
	OReK-PPC-Int	OReK-PPC-*Cached*	
Salvaguarda e Restauro do Contexto	$1,4\mu s$	$0,8\mu s$	Qualquer
Selecção da Próxima Tarefa	$0,5\mu s$	$0,5\mu s$	
Mudança de Contexto da Tarefa(Total)	$1,9\mu s$	$1,3\mu s$	Qualquer

Tabela 4.3: Sistema Sem Co-Processador - Mudança de Contexto da Tarefa

Na tabela 4.2, são apresentados os resultados temporais obtidos para o sistema com a presença do co-processador para duas situações, troca de contexto do *background* (**BG**) para uma tarefa e de uma tarefa para outra tarefa, sendo que na tabela 4.3 apresentados os resultados retirados de [dS08].

Os valores obtidos para o sistema OReK-CP quando comparados com os sistemas OReK-PPC-Int e OReK-PPC-*Cached*, são idênticos apenas na situação em que o sistema utiliza a *cache* do processador. Verificou-se no entanto, que para o sistema sem o uso da *cache*, houve uma forte degradação do desempenho, que no entanto é explicável devido ao facto de nos sistemas OReK-PPC-Int e OReK-PPC-*Cached*, ser utilizado o canal de acesso dedicado à memória do sistema, o que não acontece em OReK-CP. Só este facto sozinho, explica o porquê de existir uma degradação do desempenho.

4.4.3 Temporização da Terminação de Tarefa

A primitiva de terminação de tarefa é utilizada para indicar que a tarefa que se encontra no contexto do processador, irá terminar a sua execução e passar o seu estado para *idle*. A execução desta primitiva é feita imediatamente a seguir à terminação da função *main* do código pertencente à tarefa.

Comparando com a primitiva de mudança de contexto, quando a primitiva de terminação de tarefa é executada, ainda não se encontra determinada qual a próxima tarefa a executar, o que é necessário ordenar ao co-processador para executar o escalonador.

Assim sendo, as acções da primitiva de terminação de tarefa são:

- Remoção da memória alocada para a tarefa, da memória do sistema.
- Instrução do co-processador para terminar a execução da actual tarefa, verificar se o *deadline* foi cumprido e executar o escalonador.
- Remoção da tarefa do sistema, caso esteja indicada para destruição.
- Carregamento do contexto da nova tarefa ou do *background* (**BG**), para o processador.

Nas tabelas 4.4 e 4.5 são apresentados respectivamente, os resultados para os sistemas com e sem o co-processador. Apesar da implementação OReK-CP necessitar de aceder ao co-processador, pode-se observar que mesmo assim, obtém resultados melhores que as implementações em *software*.

Parâmetro Avaliado		Implementação OReK-CP		**# Tarefas**
		Sem *Cache*	Com *Cache*	
Terminação da Tarefa	Tarefa→BG	$6,722\mu s$	$2,237\mu s$	4
	Tarefa→Tarefa	$6,959\mu s$	$2,257\mu s$	
	Tarefa→BG	$6,722\mu s$	$2,237\mu s$	8
	Tarefa→Tarefa	$6,959\mu s$	$2,257\mu s$	
	Tarefa→BG	$6,791\mu s$	$2,296\mu s$	16
	Tarefa→Tarefa	$7,029\mu s$	$2,316\mu s$	

Tabela 4.4: Sistema Com Co-Processador - Terminação da Tarefa

Parâmetro Avaliado	Implementação OReK (*Software*)		**# Tarefas**
	OReK-PPC-Int	OReK-PPC-*Cached*	
Terminação da Tarefa	$2,6\mu s$	$2,3\mu s$	4
	$3,0\mu s$	$2,7\mu s$	8
	$3,8\mu s$	$3,3\mu s$	16
Salvaguarda e Restauro do Contexto	$1,4\mu s$	$0,8\mu s$	Qualquer
Primitiva de Terminação da Tarefa(Total)	$4,0\mu s$	$3,1\mu s$	4
	$4,4\mu s$	$3,5\mu s$	8
	$5,2\mu s$	$4,1\mu s$	16

Tabela 4.5: Sistema Sem Co-Processador - Terminação da Tarefa

4.4.4 Temporização das Activações Externas de Tarefas

Uma das funcionalidades implementadas pelo co-processador que já foram descritas na secção 3.2.4, passa pela activação externa de tarefas. Esta activação é feita totalmente por *hardware*, não existindo qualquer componente implementada em *software*. O co-processador trata cada linha de interrupção de forma paralela e sendo assim, são feitas as seguintes acções:

- Detecção do pedido de activação.
- Análise temporal para verificação do intervalo mínimo entre activações.
- Activação da tarefa com pedido em paralelo da execução do escalonador.

Rotinas Internas do Co-Processador	Ciclos(a 100Mhz)	Tempo de Execução
Activação Externa de uma Tarefa	5	$50ns$

Tabela 4.6: Sistema com Co-Processador - Activação Externa de uma Tarefa

No entanto, para os sistemas OReK-PPC-Int e OReK-PPC-*Cached*, esta funcionalidade está implementada em *software*, sendo que por cada vez que há um pedido externo para activação de uma tarefa, é necessário interromper o processador e executar as rotinas internas de tratamento do executivo. Por *software*, são executadas as seguintes acções:

- Salvaguarda do contexto da tarefa interrompida.
- Invocação da primitiva *IntCallback*

– Activação da tarefa aperiódica correspondente à interrupção gerada.

- Restauro do contexto da tarefa anteriormente interrompida.

Parâmetro Avaliado	Implementação OReK (*Software*)		**# Tarefas**
	OReK-PPC-Int	OReK-PPC-*Cached*	
Salvaguarda e Restauro do Contexto	$1,4\mu s$	$0,8\mu s$	Qualquer
Latência intermédia entre a salvaguarda do contexto e primitiva de tratamento de interrupção.	$1,2\mu s$	$1,6\mu s$	
Activação da Tarefa Aperiódica	$0,2\mu s$	$0,2\mu s$	
Latência intermédia entre a primitiva de tratamento de interrupção e o restauro do contexto.	$2,2\mu s$	$2,1\mu s$	
Pedido de Activação de uma Tarefa(Total)	$5\mu s$	$4,7\mu s$	Qualquer

Tabela 4.7: Sistema Sem Co-Processador - Activação Externa de uma Tarefa

Nas tabelas 4.6 e 4.7, estão os resultados para o sistema com e sem a utilização do co-processador, respectivamente. Comparando os resultados, pode ser observado que existe uma clara melhoria com a utilização do co-processador. Ainda, com a utilização do co-processador, é possível fazer a activação em paralelo de várias tarefas, ao passo que com a implementação exclusiva em *software*, a activação das tarefas é feita de forma sequencial o que prejudica a execução das tarefas do sistema, podendo ocorrer a perda de *deadline's*.

4.4.5 Temporização das Activações Periódicas

As activações periódicas, tal como já foi descrito na secção 3.2.5.2, são geridas por uma unidade de controlo. Esta unidade de controlo, implementada para cada tarefa de forma paralela, permite que as activações sejam feitas por *hardware*. É de relembrar que o pedido de execução do escalonador já é feito também em paralelo. Sendo assim, por cada tarefa periódica, são executadas as seguintes acções:

- Verificação temporal do período da tarefa.
- Activação da tarefa periódica.

Rotinas Internas do Co-Processador	Cycles	Time
Activação Periódica de uma Tarefa	2	20*ns*

Tabela 4.8: Activação Periódica de uma Tarefa

Mais uma vez, uma activação periódica de uma tarefa para os sistemas OReK-PPC-Int e OReK-PPC-*Cached*, necessita também de interromper o processador. Há no entanto uma questão que deve ser referida, pois durante o processamento periódico, poderão ser feitas

activações de tarefas aperiódicas o que vai para além de uma activação periódica de uma tarefa. Assim sendo, o processamento periódico consiste nas seguintes acções:

- Salvaguarda do contexto da tarefa interrompida.
- Rotina de processamento periódico.
  - Incremento do valor do temporizador do sistema.
  - Actualização das prioridades e contadores temporais das tarefas.
  - Manutenção das listas de tarefas *idle, ready* e *preempted*.
  - Activação das tarefas periódicas de acordo com os seus parâmetros temporais e respectivos contadores.
  - Activação das tarefas aperiódicas de acordo com as suas restrições temporais e pedidos de activação explícitos ou interrupções geradas.
  - Detecção de violações temporais.
- Restauro do contexto da tarefa anteriormente interrompida.

Parâmetro Avaliado	Implementação OReK (*Software*)		**# Tarefas**
	OReK-PPC-Int	OReK-PPC-*Cached*	
Salvaguarda e Restauro do Contexto	$1,4\mu s$	$0,8\mu s$	Qualquer
Latência intermédia entre a salvaguarda do contexto e primitiva activação de tarefas.	$0,9\mu s$	$1,3\mu s$	
Processamento Periódico	$3,3\mu s$ $3,8\mu s$ $4,7\mu s$	$3,0\mu s$ $3,5\mu s$ $4,5\mu s$	4 8 16
Latência intermédia entre a primitiva de tratamento de interrupção e o restauro do contexto.	$1,7\mu s$	$1,6\mu s$	Qualquer
Activação Periódica de uma Tarefa (Total)	$7,3\mu s$ $7,8\mu s$ $8,7\mu s$	$6,7\mu s$ $7,2\mu s$ $8,2\mu s$	4 8 16

Tabela 4.9: Sistema Sem Co-Processador - Activação Periódica de uma Tarefa

Demonstra-se através da comparação dos resultados, que a utilização do co-processador não só simplifica o processo de activação, como também é muito mais rápido e determinístico na sua execução, sendo que como a sua implementação como é feita em *hardware*, não existe qualquer utilização do processador, tornando ainda as activações periódicas independentes do número de tarefas suportadas.

4.4.6 Temporização das Primitivas do Executivo

Nesta secção são analisadas as primitivas do executivo OReK. Estas primitivas consistem principalmente em operações de gestão de interrupções, gestão de tarefas, gestão do temporizador e ainda controlo sobre a preempção de tarefas. Para a versão OReK-CP, foram analisadas todas as primitivas do executivo, no entanto em [dS08], o autor apenas apresentou uma análise para algumas primitivas nas versões OReK-PPC-Int e OReK-PPC-*Cached*. Sendo assim, só será possível apresentar uma comparação para algumas das primitivas.

Primitiva Avaliada	Implementação OReK-CP		**#**
	Sem *Cache*	Com *Cache*	**Tarefas**
Inicialização do Executivo	$30,808\mu s$	$18,810\mu s$	4
	$32,234\mu s$	$19,641\mu s$	8
	$34,838\mu s$	$21,423\mu s$	16
Terminação do Executivo	$27,297\mu s$	$13,305\mu s$	4
	$40,286\mu s$	$19,503\mu s$	8
	$65,567\mu s$	$30,765\mu s$	16
Arranque do Executivo	$1,227\mu s$	$0,752\mu s$	Qualquer
Paragem do Executivo	$0,475\mu s$	$0,376\mu s$	
Inibição de Interrupções	$0,227\mu s$	$0,221\mu s$	
Activação de Interrupções	$0,227\mu s$	$0,221\mu s$	
Lêr Contagem de *Ticks*	$1,283\mu s$	$0,531\mu s$	
Inibição da Preempção	$0,498\mu s$	$0,353\mu s$	
Activação da Preempção	$0,498\mu s$	$0,372\mu s$	
Criar Tarefa Ordinária	$10,470\mu s$	$5,187\mu s$	
Criar Tarefa Periódica(*Soft*)	$10,530\mu s$	$5,128\mu s$	
Criar Tarefa Aperiódica(*Soft*)	$12,668\mu s$	$7,999\mu s$	
Criar Tarefa Periódica(*Hard*)	$10,589\mu s$	$6,157\mu s$	
Criar Tarefa Aperiódica(*Hard*)	$12,688\mu s$	$8,038\mu s$	
Destruir Tarefa	$6,553\mu s$	$3,742\mu s$	
Iniciar Tarefa	$11,206\mu s$	$5,563\mu s$	
Parar Tarefa	$3,286\mu s$	$1,940\mu s$	
Activar Tarefa	$2,633\mu s$	$1,564\mu s$	
Lêr Estado da Tarefa	$3,069\mu s$	$1,980\mu s$	

Tabela 4.10: Implementação Com Co-Processador - Primitivas do Executivo

Nas tabelas 4.10 e 4.11, apresenta-se a avaliação temporal para os sistemas com e sem utilização do co-processsador, respectivamente.

Comparando os resultados, podemos observar que em termos de execução das primitivas, os sistemas que não utilizam o co-processador, obtêm resultados substancialmente melhores. A razão pela qual os resultados no sistema OReK-CP são tão drásticos, deve-se à latência inserida pelo barramento local como demonstram os resultados na secção 4.4.1, contendo também a solução para o problema. Como as primitivas do executivo OReK na versão OReK-CP interagem constantemente com o co-processador, o tempo de execução é fortemente afectado prejudicando assim o desempenho das primitivas.

No entanto, mesmo apesar dos resultados de desempenho serem piores, existe um ganho

Primitiva Avaliada	Implementação OReK (*Software*)		**#**
	OReK-PPC-Int	OReK-PPC-*Cached*	**Tarefas**
Inicialização do Executivo	$24,9\mu s$	$26,0\mu s$	4
	$26,5\mu s$	$28,7\mu s$	8
	$29,4\mu s$	$34,1\mu s$	16
Inibição da Preempção	$0,1\mu s$	$0,1\mu s$	Qualquer
Activação da Preempção	$0,1\mu s$	$0,1\mu s$	
Criar Tarefa	$1,8\mu s$	$1,8\mu s$	
Destruir Tarefa	$3,2\mu s$	$3,0\mu s$	
Iniciar Tarefa	$4,7\mu s$	$5,0\mu s$	4
	$5,2\mu s$	$5,4\mu s$	8
	$6,3\mu s$	$6,4\mu s$	16
Parar Tarefa	$3,0\mu s$	$4,0\mu s$	Qualquer
Activar Tarefa Aperiódica	$2,0\mu s$	$1,8\mu s$	4
	$2,5\mu s$	$2,3\mu s$	8
	$3,6\mu s$	$3,2\mu s$	16
Lêr Estado da Tarefa	$0,7\mu s$	$1,0\mu s$	Qualquer

Tabela 4.11: Implementação Sem Co-Processador - Primitivas do Executivo

de grande importância. A vantagem obtida pela utilização do co-processador, é o facto de a execução das primitivas do executivo, com excepção das primitivas de *Inicialização do Executivo* e *Terminação do Executivo*, serem independentes do número de tarefas suportadas pelo sistema. Este facto é de especial importância, quando se tem em conta as instruções de maior uso como por exemplo, *Inibição e Activação de Interrupção, Criar Tarefa, Destruir Tarefa, Activar Tarefa e Lêr Estado da Tarefa*. Como o objectivo principal de um sistema de tempo-real é ser temporalmente preciso na execução das suas tarefas, o determinismo destas primitivas, ganha uma elevada importância pois ao não estarem dependentes do número de tarefas existentes e do estado de execução das mesmas, o seu tempo de execução torna-se fixo, eliminando-se assim, possíveis fontes de indeterminismo do sistema.

4.4.7 Temporização dos Semáforos

Nesta secção, é feita a análise temporal das primitivas de controlo dos semáforos, implementados pelo co-processador.

As funcionalidades implementadas são idênticas às versões dos sistemas OReK-PPC-Int OReK-PPC-*Cached*, sendo apenas a diferença encontrada na primitiva *Activar Semáforo*, que não tem implementação no co-processador devido à própria natureza da implementação dos semáforos em *hardware*, estando todos os semáforos sempre activos. A razão pela qual os semáforos estão sempre activos, acontece por causa de não existir noção de alocação de semáforos no co-processador, pois no momento de arranque do sistema, estes já existem com um nível de preempção pré-definido (ver capítulo 3 - secção 3.2.6).

As tabelas 4.12 e 4.13, mostram os resultados temporais da execução das primitivas de controlo dos semáforos, para as implementações com co-processador e sem co-processador, respectivamente.

Tal como foi explicado na secção 4.4.1 e já verificado na análise das primitivas principais

Primitiva Avaliada	Implementação OReK-CP		# Semáforos
	Sem *Cache*	Com *Cache*	
Criar Semáforo	$7,266\mu s$	$3,722\mu s$	Qualquer
Destruir Semáforo	$6,296\mu s$	$3,362\mu s$	
Registar Tarefa em Semáforo	$4,039\mu s$	$2,296\mu s$	
Activar Semáforo	Não Aplicável		
Fechar Semáforo	$3,326\mu s$	$1,821\mu s$	Qualquer
Libertar Semáforo	$3,148\mu s$	$1,603\mu s$	

Tabela 4.12: Implementação Com Co-Processador - Primitivas dos Semáforos

Primitiva Avaliada	Implementação OReK-CP		# Semáforos
	OReK-PPC-Int	OReK-PPC-*Cached*	
Criar Semáforo	$0,8\mu s$	$0,8\mu s$	Qualquer
Destruir Semáforo	$0,8\mu s$	$0,1\mu s$	
Registar Tarefa em Semáforo	$1,1\mu s$	$1,0\mu s$	
Activar Semáforo	$0,7\mu s$	$0,6\mu s$	
Fechar Semáforo	$0,9\mu s$	$0,8\mu s$	
Libertar Semáforo	$0,8\mu s$	$0,7\mu s$	

Tabela 4.13: Implementação Sem Co-Processador - Primitivas dos Semáforos

do executivo, a latência provocada pelo barramento local, tem um forte impacto no tempo de execução das primitivas, na implementação OReK-CP. Os resultados mostram claramente, que são os sistemas OReK-PPC-Int OReK-PPC-*Cached* com os melhores tempos de execução, sendo a razão do sucedido a mesma das primitivas principais, que é devido ao facto de o acesso à memória ser dedicado.

4.5 Recursos da FPGA Usados

Como a implementação do co-processador, está dependente dos recursos disponíveis nos circuitos reconfiguráveis, foram feitas várias sintetizes de circuitos com o intuito de mostrar os recursos que o co-processador consome. Na tabela 4.14, são apresentados os recursos consumidos exclusivamente pelo co-processador para uma configuração com 5 linhas de interrupções externas e 4 semáforos, sendo indicado a percentagem de recursos necessários em relação à FPGA Virtex 2 Pro (XUPV2Pro). Não estão incluídos os recursos consumidos pelo barramento local e pela memória do sistema.

Note-se que apesar de não ser possível testar o executivo para 32 tarefas devido à falta de recursos, foi apresentado na mesma a quantidade de recursos necessários para a sua implementação, sendo os valores de 64, 128 e 256 deixados de parte. Isto acontece devido a ser necessário implementar um escalonador manualmente, para cada parâmetro diferente do número de tarefas, que é um resultado da limitação do VHDL.

É no entanto possível ter uma ideia dos recursos necessários, visto que à medida que o número de tarefas vai dobrando, os recursos necessários acompanham a mesma proporção

Número de Tarefas Suportadas	*Flip-Flops*		*LUT's*		*Slices*	
4	3243	12,8%	5943	21,7%	3302	24,1%
8	4673	17,0%	9623	35,1%	5137	37,5%
16	7529	27,5%	16805	61,3%	8945	65,3%
32	12303	44,9%	30765	112,3%	16307	119%

Tabela 4.14: Utilização de Recursos por parte do Co-Processador para 5 Linhas de Interrupções e 4 Semáforos

de forma aproximada.

Capítulo 5

Conclusão

Este capítulo termina a dissertação, com um resumo dos objectivos atingidos, sendo feito um resumo final sobre os resultados obtidos.

São também apresentados futuros desenvolvimentos do projecto que serão possíveis que poderão trazer um suporte mais completo ao co-processador.

5.1 Resumo da implementação

O principal objectivo desta dissertação, era a construção de um co-processador que desse suporte por *hardware*, às diversas funcionalidades internas do executivo OReK. As funcionalidades suportadas pelo co-processador são:

- Gestão de tarefas (periódicas, aperiódicas e ordinárias):
  - Implementação dos atributos de cada tarefa, com acesso em paralelo.
  - Implementação de um gestor de activação periódica para cada tarefa.
- Gestão de linhas de interrupção externas:
  - Um detector de pedidos de activação por cada linha.
  - Um analisador de intervalo mínimo entre activações por cada linha.
- Escalonamento de tarefas segundo três possíveis algoritmos (*Rate Monotonic, Deadline Monotonic e Earliest Deadline First*).
- Implementação de um *dispatcher* para gestão da interrupção do processador, para mudança de contexto.
- Gestão de semáforos (utilização da política de pilha de recursos).

Com a implementação do co-processador conseguiu-se:

- Reduzir o *overhead* do executivo (eliminação das interrupções periódicas associadas ao temporizador).
- Tornar o tempo de execução das primitivas e rotinas internas do executivo, previsível e fixo (independente do estado das tarefas).
- Filtrar os pedidos de interrupção externos, através do estabelecimento de um intervalo mínimo entre pedidos.
- Paralelizar a activação de tarefas periódicas e aperiódicas, sendo o tempo da activação fixo.

5.2 Análise final dos resultados

Como foi demonstrado no capítulo 4, a utilização do co-processador tem um forte impacto no melhoramento do determinismo do executivo. No entanto, durante a análise, foram muitos os casos em que a execução de primitivas que utilizam o co-processador durante a execução, são penalizadas com uma elevada latência, impacto causado pelo barramento partilhado.

5.2.1 Benefícios

A utilização do co-processador, possibilitou que diversas tarefas do executivo OReK pudessem ser implementadas em *hardware* tirando partido do paralelismo disponível. É na gestão da activação das tarefas periódicas e na gestão das linhas de interrupção externas que o impacto do paralelismo se verifica mais.

A gestão de forma paralela de cada linha de interrupção externa (secção 3.2.4), permitiu que cada uma fosse gerida de forma paralela e independente, sendo os próprios pedidos de activação de cada periférico externo, analisados com o fim de impedir que tarefas sejam activadas antes do intervalo mínimo entre activações ser cumprido.

A implementação de um controlador de activação para cada tarefa (secção 3.2.5), permite que as activações sejam feitas de forma simultânea e independentes, reduzindo a latência entre o *tick* de activação e o momento exacto em que a tarefa é colocada em execução.

O escalonamento das tarefas através do co-processador (secção 3.2.5.3), permite que se consiga obter um resultado muito mais rápido do que a sua versão implementada em *software*, sendo que o *dispatcher* (secção 3.2.5.4) impede que o processador seja interrompido, não existindo assim interrupção do processador.

Por fim, o gestor de semáforos (secção 3.2.6), possibilita a gestão do acesso a recursos partilhados, impedindo a execução de tarefas que não têm uma prioridade adequada, o que também previne a interrupção desnecessária do processador e ainda, previne problemas ligados ao acesso a recursos partilhados.

A implementação de todos os gestores e controladores indicados anteriormente, permitiu a remoção de grande parte da carga do executivo do processador. Estes foram as principais acções, que deixaram de ser executadas pelo processador:

- Activações periódicas para gestão das activações periódicas e das tarefas aperiódicas sujeitas ao intervalo mínimo entre activações.
- O escalonamento de tarefas.
- Atendimento de pedidos de interrupção externos.

Assim, com a utilização do co-processador, a carga de ocupação do executivo tornou-se praticamente nula, sendo apenas provocada pela mudança de contexto e terminação de tarefa que são dependentes da arquitectura do processador presente no sistema, sendo no entanto, ambas as situações completamente determinísticas.

5.2.2 Custos

A latência de acesso ao co-processador introduzida pelo barramento local, tem um forte impacto nas primitivas do executivo que usam o co-processador. Um ponto negativo que foi detectado na avaliação do co-processador, foi precisamente a elevada latência que o barramento local gerava quando era acedido, quer para lêr, quer para escrever.

No entanto, esta latência pode ser diminuída ou até mesmo removida. Caso o processador fornecesse um interface de acesso a uma unidade de processamento auxiliar, o co-processador poderia ser adaptado com alterações mínimas e interligado ao processador através deste canal de comunicação dedicado.

Não se pode no entanto descurar, este problema colocado com a latência de acesso. Caso a latência seja demasiado elevada, os acessos ao co-processador podem colocar em causa, os benefícios obtidos com a utilização do mesmo! Sendo assim, este é um factor que deve ser tido em conta.

Outro problema que depende do circuito reconfigurável, é o facto da parametrização do co-processador estar directamente ligada aos recursos disponíveis. Caso o circuito reconfigurável possua recursos muito limitados, o número de tarefas, número de linhas de interrupção externas ou mesmo o número de semáforos, poderão ser muito limitados, podendo resultar numa implementação do co-processador com um suporte insuficientes para correr todas as tarefas necessárias, visto que esta arquitectura explora o paralelismo das operações sempre que possível, o que provoca uma elevada necessidade de recursos lógicos.

Para finalizar, deve também ser referido que a necessidade do co-processador suportar mais tarefas, mais linhas de interrupção externas ou mais semáforos, aumenta a complexidade da arquitectura e consequentemente, aumenta de forma drástica o tempo de síntese do circuito final. Dando como exemplo, uma arquitectura com suporte para 16 tarefas, 5 linhas de interrupção e 4 semáforos, juntamente com o barramento local mais memórias e dispositivos lógicos simples do sistema, leva aproximadamente 20 minutos a sintetizar utilizando um processador Intel Core 2 Duo P8400 a 2,2Ghz!

5.3 Futuros Desenvolvimentos

Apesar da implementação deste co-processador genérico, ser uma inovação para o executivo OReK, existem ainda áreas por explorar. Sendo assim, os seguintes aspectos poderão ser desenvolvidos em futuros projectos:

- Ligação ponto-a-ponto entre o CPU e o co-processador (ligação dedicada).
  - Justifica-se a implementação de uma ligação dedicada entre o processador do sistema e o co-processador, pois como se pode observar na avaliação do co-processador (ver capítulo 4), o acesso ao co-processador é penalizado com a inserção de latência por parte do barramento partilhado. Uma ligação ponto-a-ponto, iria permitir um acesso directo ao co-processador, sendo eliminada a latência introduzida pelas esperas no acesso ao barramento local.
- Execução das instruções do co-processador em paralelo com as instruções do processador (co-processador não bloqueante).

– Como foi explicado no capítulo 3, optou-se por tornar as operações do co-processador atómicas, devido à complexidade da arquitectura. No entanto, poderia-se tirar partido da execução em paralelo de instruções por parte de ambas as unidades, com o intuito de executar acções em paralelo dando como exemplo, na terminação de uma tarefa, a possibilidade de executar em paralelo, o escalonador e a remoção da memória, da *stack* da tarefa.

5.4 Conclusão final

Com esta dissertação, concluí-se que a utilização de um co-processador implementado num sistema reconfigurável, é uma mais valia para a execução do sistema de tempo-real. Ficou provado que a utilização do co-processador, não só elimina fontes de indeterminismo como também estabelece um tempo de execução fixo para as próprias primitivas e operações internas do executivo.

O co-processador permitiu que a maior parte das funcionalidades internas do executivo OReK, passem agora a ser executadas em *hardware*, reduzindo a carga do executivo sobre o processador. As interrupções periódicas geradas pelo temporizador interno do processador, para se proceder à gestão das activações de tarefas, é implementado pelo co-processador, sendo apenas feita a interrupção do processador, quando o escalonador em conjunto com o *dispatcher* determina que é necessário uma troca de contexto de tarefas. A gestão dos pedidos de interrupção por parte de periféricos externos, fica também a cargo do co-processador, evitando-se assim, a interrupção do processador para proceder à análise do pedido.

A implementação em hardware das operações internas do executivo, permite libertar tempo de processamento que poderá ser entregue às tarefas, sendo reduzido o *overhead* causado pela execução das mesmas em *software*, a possibilidade de perdas de *deadline* por parte das tarefas e ter a possibilidade do processador operar a frequências inferiores.

Ainda, visto que o co-processador foi construído a pensar na sua utilização com um processador genérico, permite que a sua utilização seja alargada para suportar outros processadores. Estamos na presença de um co-processador escalável a outros sistemas.

No anexo seguinte, está apresentado de forma detalhada, a constituição do executivo OReK.

Apêndice A

O Executivo de Tempo-real OReK

Sumário
Este anexo apresenta o executivo *OReK* (*Object-oriented Real-time Kernel*). Começa por uma breve introdução, seguida da exposição das suas funcionalidades, arquitectura interna e implementação. Por fim, é descrita a sua utilização em aplicações, sendo exibido um exemplo concreto.

A informação apresentada neste anexo foi baseada em [ASRdO07] e em [dS08]. Com o consentimento e a pedido de ambos os autores, a informação aí contida foi estendida de forma a construir um documento coerente que inclua a descrição das novas funcionalidades e plataformas suportadas pelo executivo *OReK*.

A.1 Introdução

O *OReK* é um executivo de tempo-real orientado por objectos completamente preemptivo e implementado maioritariamente em *C++*.

Tal como já foi mencionado no capítulo 2.5 - secção 2.1, um executivo é uma entidade ou um módulo de *software* responsável pela execução concorrente de tarefas ou processos. As funções normalmente realizadas por um executivo são o escalonamento, o lançamento em execução, a comutação, a terminação, a comunicação e a sincronização de tarefas. Para as duas últimas funções e para controlar o acesso a recursos partilhados, os executivos disponibilizam tipicamente primitivas do tipo semáforos, eventos, mensagens, ou outros com funcionalidades análogas.

Em sistemas simples é comum integrar num único módulo executável, isto é, de forma monolítica, o executivo e as tarefas que constituem o sistema. Em sistemas mais complexos que necessitem, por exemplo, de carregamento dinâmico de tarefas ou processos, de gestão hierárquica de memória, de serviços de rede ou de dispositivos de entrada/saída sofisticados é usual utilizar-se um sistema operativo, do qual o executivo é uma das peças fundamentais.

Um sistema de tempo-real é normalmente um sistema de controlo reactivo que responde continuamente a eventos produzidos pelo ambiente em que está inserido. A resposta é feita de acordo com uma estratégia predefinida e cumprindo as restrições temporais definidas. A execução do sistema é despoletada por eventos que podem ser síncronos (periódicos) ou assíncronos (aperiódicos) e provenientes de várias fontes, tais como um temporizador ou

um sensor. Em qualquer dos casos, a resposta ao evento é feita através da execução de uma tarefa ou processo, onde o evento é processado e desencadeada a respectiva reacção.

Um executivo de tempo-real é um executivo capaz de gerir tarefas de tempo-real, isto é, com restrições temporais precisas, destinando-se portanto a sistemas de tempo-real. Um executivo de tempo-real tem a responsabilidade de executar as tarefas cumprindo as restrições temporais do sistema. A utilização de executivos em sistemas de tempo-real para fazer a gestão de tarefas tem a vantagem de simplificar o desenvolvimento, de os tornar mais robustos e de proporcionar uma abstracção do *hardware*, tornando-os independentes da plataforma, promovendo assim a portabilidade.

O executivo *OReK* descrito neste capítulo teve a sua origem no executivo *ReTMiK* [AGP03]. Relativamente ao *ReTMiK*, foram várias as alterações e melhoramentos introduzidos, nomeadamente:

- A conversão do código fonte escrito em linguagem *C* para *C++* e adopção do paradigma de orientação por objectos.
- A gestão de tarefas com base em listas bi-ligadas de forma a optimizar a sua manipulação.
- O suporte para conjuntos heterogéneos de tarefas incluindo as de tempo-real críticas e não críticas, quer periódicas quer aperiódicas e tarefas ordinárias de baixa prioridade.
- A capacidade de gestão de tarefas em grupos definidos pela aplicação.
- A adição de primitivas de controlo da preempção das tarefas e de sincronização baseada em semáforos com limitação do tempo de bloqueio em que ocorrem fenómenos de inversão de prioridade nas tarefas em execução.
- A inclusão dos mecanismos que permitam tirar partido das capacidades de multi-tarefa simultânea do processador *ARPA-MT* [ASRdO07].
- O suporte de utilização do co-processador genérico OReK-CP, para aceleração e melhoramento do determinismo do executivo OReK.

A.1.1 Plataformas Suportadas

A generalidade do executivo *OReK* é independente do processador alvo, possuindo apenas segmentos bem localizados de código específico da plataforma. Actualmente pode ser utilizado em *PCs* com processador compatível com a família *Intel x86*, no processador *ARPA-MT* e nos sistemas integrados *MB-SoC* e *PPC-SoC*, os quais incluem os processadores *MicroBlaze* e *PowerPC 405*, respectivamente.

No primeiro caso, o executivo *OReK* deve ser executado directamente sobre *MSDOS*, uma vez que necessita de configurar e aceder ao *hardware* do *PC*, mais concretamente ao temporizador e ao controlador de interrupções. No caso da plataforma *ARPA-MT*, o executivo *OReK* pode funcionar total ou parcialmente em *software*, sendo na segunda hipótese suportado pelo co-processador *Cop2-OSC* (*Cop2 - Operating System Coprocessor*) [ASRdO07]. Por último, no caso dos sistemas integrados *MB-SoC* e *PPC-SoC*, o executivo *OReK* poderá operar completamente em *software* ou então, poderá recorrer ao suporte em *hardware* para aceleração das suas funcionalidades, utilizando o co-processador genérico OReK-CP.

O executivo *OReK* implementa uma camada de abstracção, fornecendo um conjunto de serviços para gestão de tarefas e semáforos com restrições temporais através de uma interface independente da plataforma utilizada (inclusivamente da implementação do co-processador *Cop2-OSC*) [ASRdO07]. Se este co-processador estiver implementado, o executivo *OReK* usa as funcionalidades por ele disponibilizadas, caso contrário executa todas as suas funções completamente em *software*.

O executivo *OReK* foi desenvolvido em *C++*, fornecendo uma interface de programação orientada por objectos. É disponibilizado na forma de uma biblioteca destinada a ser ligada com o código da aplicação, obtendo-se no final do fluxo de compilação um módulo monolítico, contendo o código máquina e os dados quer da aplicação quer do executivo.

Na realidade são disponibilizadas três bibliotecas, duas correspondendo às implementações no processador *ARPA-MT* descrito em [ASRdO07] e uma para as implementações nas plataformas *MB-SoC* e *PPC-SoC*, descritas em [dS08]. No primeiro caso as duas implementações distinguem-se pelo suporte ou não do co-processador *Cop2-OSC*, designadas por *OReK-ARPA-C2* e *OReK-ARPA-Sw*, respectivamente. Por último, na implementação do executivo *OReK* nos processadores *MicroBlaze* e *PowerPC 405*, designados por *OReK-MB* e *OReK-PPC*, respectivamente, podendo actualmente para o processador PowerPC 405 ser utilizado o co-processador genérico OReK-CP para aceleração das funcionalidades internas do executivo.

Todas as bibliotecas implementam a mesma interface. Desta forma é possível utilizar o mesmo código fonte da aplicação independentemente da plataforma.

A.1.2 Características Gerais

No estado actual, o executivo *OReK* fornece serviços de temporização, gestão de tarefas e sincronização no acesso a recursos partilhados com base em semáforos binários e primitivas de controlo de preempção das tarefas. Na implementação destinada ao processador *ARPA-MT* é também capaz de gerir os diversos contextos de execução, podendo controlar as capacidades de multi-tarefa simultânea do processador.

As resoluções temporais permitidas dependem da plataforma base e da dimensão do conjunto de tarefas, o qual define os requisitos computacionais das funções internas do executivo. Com o suporte do co-processador *Cop2-OSC* podem ser alcançados valores tão pequenos quanto um micro segundo.

Em termos genéricos, os serviços disponibilizados pelo executivo *OReK* dividem-se nos seguintes grupos:

- **Configuração e estado** (arranque, terminação e informação temporal).
- **Diagnóstico** (códigos de erro e mensagens).
- **Gestão de tarefas** (criação, destruição, manipulação, etc.).
- **Gestão de semáforos** (criação, destruição, manipulação, etc.).

Os tipos de tarefas considerados na implementação do executivo *OReK* foram os seguintes:

- **Ordinárias** (*non real-time*).
- **Periódicas de tempo-real não críticas** (*soft real-time periodic*) ou simplesmente periódicas.

- **Aperiódicas de tempo-real não críticas** (*soft real-time aperiodic*) ou simplesmente aperiódicas.
- **Periódicas de tempo-real críticas** (*hard real-time periodic*) ou simplesmente periódicas de tempo-real.
- **Aperiódicas de tempo-real críticas** (*hard real-time aperiodic*) ou simplesmente aperiódicas de tempo-real.

A tabela A.1 apresenta os parâmetros suportados pelas tarefas do executivo *OReK*. Além destes, existe ainda a possibilidade de especificar o contexto de execução de cada tarefa. Porém, apenas o processador *ARPA-MT* suporta vários contextos de execução [ASRdO07], pelo que, para as plataformas *MB-SoC* e *PPC-SoC*, todas as tarefas terão o mesmo contexto de execução.

Tarefas	Prioridade base	Execução única	Identificação do grupo	Tamanho da pilha	Período	Fase inicial	*MIT*	Ligação a interrupção externa	*Deadline* relativa
Ordinárias	✓	✓	✓	✓					
Periódicas			✓	✓	✓	✓			
Aperiódicas			✓	✓			✓	✓	
Periódicas de tempo-real			✓	✓	✓	✓			✓
Aperiódicas de tempo-real			✓	✓			✓	✓	✓

Tabela A.1: Quadro resumo dos parâmetros suportados pelas tarefas do executivo *OReK*.

Dentro de cada contexto de execução, o escalonamento das tarefas é feito segundo os critérios *EDF* (*Earliest Deadline First*), *RM* (*Rate Monotonic*), *DM* (*Deadline Monotonic*) e *FIFO* (*First Come-First Served*) para as tarefas de tempo-real críticas, tempo-real não críticas e ordinárias, respectivamente.

Os semáforos implementados no executivo *OReK* são do tipo binário, usados para garantir a exclusão mútua na entrada em regiões críticas onde é feito o acesso a recursos partilhados.

Para gerir os semáforos é usado o protocolo *SRP* (*Stack Resource Policy*), uma vez que pode ser usado em conjunto com políticas de escalonamento baseadas em prioridades estáticas (tais como a *RM* e a *FIFO*) ou dinâmicas (tal como a *EDF*). Além disso, tal como mencionado no capítulo 2.5 secção 2.1.3.2, o protocolo *SRP* limita o tempo de bloqueio de tarefas de alta prioridade devido a semáforos detidos por tarefas de menor prioridade e previne a ocorrência de bloqueios em cadeia e de *deadlocks*.

Estas funcionalidades estão implementadas completamente em *software* nas versões *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC*, e de forma híbrida em *hardware-software* para a versão OReK-CP que utiliza o co-processador genérico e ainda, na versão *OReK-ARPA-C2* com base nos serviços disponibilizados pelo *Cop2-OSC*.

No caso das implementações completas em *software*, além das funções que implementam os serviços disponibilizados à aplicação, o executivo inclui ainda as funções `TimerCallback` e `IntCallback`. A primeira deve ser invocada no contexto da rotina de serviço à interrupção do temporizador do sistema e a sua função é executar o processamento periódico, o escalonamento e a comutação da tarefa em execução. A segunda é invocada a partir da rotina de serviço às interrupções dos periféricos para activação das tarefas aperiódicas correspondentes.

Por outro lado, na implementação suportada pelo co-processador *Cop2-OSC* é disponibilizada a rotina `OSCCallback` cuja função é apenas a comutação da tarefa em execução, devendo para tal ser invocada no contexto da rotina de tratamento das excepções geradas pelo co-processador *Cop2-OSC*.

Para a implementação OReK-CP, é disponibilizada a rotina `Interruption` cujo simples objectivo, é a comutação de tarefas. Esta rotina é executada quando o co-processador genérico, gera uma interrupção indicando que existe uma tarefa com uma prioridade superior à tarefa em actual execução.

Embora com diferentes desempenhos, todas as implementações disponibilizam as mesmas funcionalidades e a mesma interface do ponto de vista da aplicação o que é vantajoso no panorama do desenvolvimento de aplicações e possibilita a comparação directa entre as implementações integrais em *software* do executivo *OReK* e uma em que as primitivas básicas são realizadas em *hardware*.

A.2 Utilização do Paradigma de Orientação por Objectos

O executivo *OReK* foi desenvolvido com a linguagem *C++*, a qual suporta o Paradigma de Orientação por Objectos (*Object Oriented Paradigm - OOP*). Este paradigma no desenvolvimento de *software* consiste na criação de aplicações constituídas por vários objectos que interagem. Cada objecto é uma estrutura de dados que integra também um conjunto de funções internas que operam sobre os campos de dados e um conjunto de funções de interface que permitem trocar informação com o exterior. A utilização do *OOP* no desenvolvimento do executivo e na implementação das tarefas foi motivada por algumas potencialidades interessantes deste paradigma, nomeadamente, a abstracção de dados, a herança, o encapsulamento e o polimorfismo. Em conjunto, estes mecanismos permitem a construção de programas mais concisos e legíveis e, facilitam a reutilização de código pré-existente. Estas facilidades podem ser descritas sumariamente da seguinte forma:

- **Abstracção de dados** - uma linguagem baseada no *OOP* permite definir novos tipos de dados (representados por classes na linguagem *C++*) e as operações que sobre eles podem ser executadas (métodos e operadores em *C++*). Esta funcionalidade, em conjunto com a capacidade de efectuar a sobrecarga (redefinição) das funções e dos operadores, permite utilizar intuitivamente os novos tipos de dados definidos pelo utilizador de forma análoga aos predefinidos na linguagem. Uma vez definida uma classe, podem-se declarar objectos dessa classe, isto é, criar instâncias da classe, da mesma forma que se declaram variáveis dos tipos predefinidos da linguagem.

- **Herança** - a partir de uma classe base podem ser derivadas uma ou mais classes. Através do mecanismo de herança é possível modificar e/ou estender, numa classe

derivada, a funcionalidade da classe base. A classe derivada pode possuir novos atributos e métodos, bem como estender e/ou redefinir os métodos da classe base, facilitando assim a reutilização de código pré-existente. O desenvolvimento é também simplificado, uma vez que tudo o que é comum a um conjunto de classes pode ser colocado numa ou várias classes base.

- **Encapsulamento** - no caso do *C++*, uma classe possui variáveis e funções, também vulgarmente designadas por atributos e métodos, respectivamente. A visibilidade ou acessibilidade dos atributos e métodos pode ser individualmente especificada. Existem três níveis de acessibilidade distintos: privado, protegido e público. Por omissão os elementos são privados, isto é, são acessíveis apenas dentro da classe em que estão definidos. Os atributos e métodos protegidos são também acessíveis nas classes derivadas. Finalmente, um elemento público é visível em qualquer parte do programa onde o objecto possa ser acedido. O encapsulamento consiste geralmente em tornar privados ou protegidos os atributos da classe e públicos os seus métodos de interface. Isto significa que o acesso aos atributos é feito a partir dos métodos da classe, o que facilita a manutenção da integridade interna do objecto.

- **Polimorfismo** - este mecanismo permite que o método invocado para um dado objecto seja determinado durante a execução do programa em função do tipo efectivo do objecto, em vez de ser determinado estaticamente pelo compilador. No *C++* o polimorfismo é implementado através de funções virtuais e de ponteiros/referências para objectos.

No executivo *OReK*, uma tarefa é um objecto implementado em *C++* através de uma classe derivada da `COReKTask`. Esta fornece um conjunto de serviços base para manipulação de tarefas, tais como criação, destruição, terminação, arranque, paragem, activação, etc. Tal como qualquer classe, uma tarefa possui pelo menos um constructor e método(s), podendo também ter um destructor e atributos. Uma classe que implementa uma tarefa pode ser instanciada diversas vezes numa aplicação correspondendo cada uma a uma tarefa distinta que executa concorrentemente com as outras tarefas do sistema. Cada instância é tipicamente configurada durante a sua criação através de parâmetros do respectivo constructor ou método de inicialização. Uma tarefa pode também possuir vários métodos específicos da aplicação. Contudo, existe um mandatório e que define o ponto de entrada da tarefa, isto é, o método invocado pelo executivo quando é iniciada a execução da tarefa. A sua assinatura, ou protótipo, deve ser o seguinte:

```
virtual void Main();
```

A classe `COReKTask` não é instanciável porque é abstracta, uma vez que o método de entrada da tarefa está definido como uma função virtual pura, isto é, não possui implementação.

Em suma, na implementação do executivo *OReK* a herança é utilizada para colocar numa classe base abstracta todas as estruturas de dados e de controlo comuns a todas as tarefas. O encapsulamento permite esconder essas estruturas. O polimorfismo simplifica a implementação do método de entrada da tarefa.

A.2.1 Tipos de Variáveis

Do ponto de vista da duração, num programa existem em geral três tipos de variáveis, correspondendo cada um a diferentes zonas ou segmentos de dados:

- **Automáticas** - declaradas no corpo das funções ou métodos. As variáveis automáticas são colocadas em registos internos do processador ou na pilha, criadas aquando da sua declaração e destruídas automaticamente quando o bloco onde foram declaradas termina.
- **Estáticas** - estas variáveis podem ser declaradas dentro ou fora das funções ou métodos, são colocadas no segmento de dados estáticos, criadas e inicializadas quando o programa é carregado em memória ou inicia a sua execução e destruídas automaticamente quando o programa termina.
- **Dinâmicas** - as variáveis deste tipo são acedidas através de ponteiros para uma zona de dados específica designada por *heap*. As variáveis dinâmicas são criadas e destruídas explicitamente durante a execução do programa usando primitivas para alocação e libertação de memória tais como as funções `malloc` e `free` do *C* ou os operadores `new` e `delete` do *C++*.

As linguagens baseadas no OOP possuem ainda outro tipo de variável já mencionado acima, os atributos.

No contexto das tarefas, as variáveis automáticas são específicas de cada invocação de uma tarefa, isto é, não são preservadas entre diferentes instâncias da mesma tarefa. As variáveis estáticas são preservadas durante toda a execução do programa e partilhadas por todas as tarefas. Os atributos da tarefa são específicos de cada tarefa e preservados entre diferentes activações da mesma tarefa. Isto permite declarar facilmente variáveis que são específicas de cada tarefa e preservadas entre activações, sem a necessidade de passar explicitamente parâmetros à função que implementa a tarefa.

A.3 Arquitectura

A figura A.1 ilustra a constituição típica de um sistema de tempo-real baseado no executivo *OReK*. O executivo cria uma camada de abstracção que esconde alguns detalhes do *hardware*, em particular os relativos à gestão dos temporizadores e interrupções e disponibiliza serviços de temporização, de gestão de tarefas e manipulação de semáforos.

Uma aplicação baseada no executivo *OReK* é monolítica, isto é, consiste num único módulo executável ou imagem binária contendo quer a implementação das tarefas da aplicação quer o executivo *OReK*.

O sistema é composto por *M* tarefas, uma de entrada (índice 0) e *M-1* da aplicação. As tarefas da aplicação são executadas de acordo com a sua prontidão e prioridade. A tarefa de entrada é executada no arranque do sistema e quando não houver qualquer outra tarefa para executar.

Cada tarefa possui código e dados. O código é partilhado por todas as instâncias da mesma classe de tarefas, enquanto os dados (atributos e variáveis automáticas) são específicos de cada tarefa.

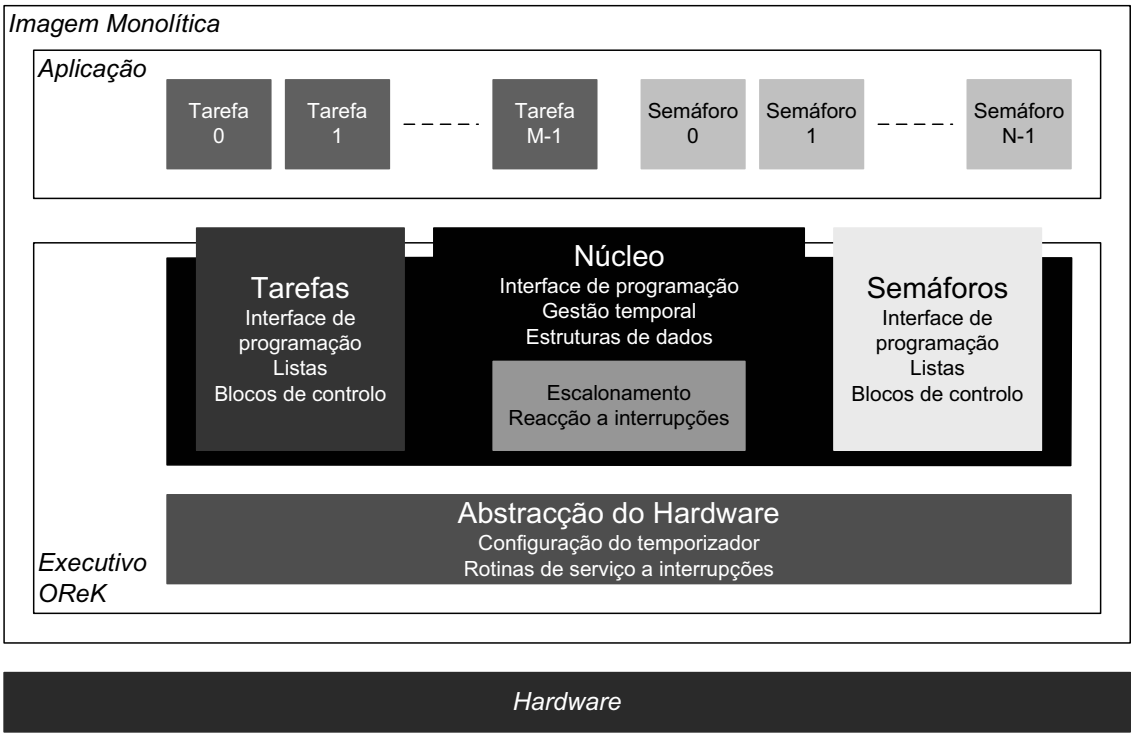


Figura A.1: Arquitectura de uma aplicação baseada no executivo *OReK* (retirado de [ASRdO07]).

O executivo *OReK* é preemptivo, isto é, permite que a execução de uma tarefa seja interrompida em qualquer instante sem que para tal a própria tarefa tenha que se auto-suspender explicitamente. Independentemente do ponto onde a tarefa se encontre, o executivo é capaz de interromper a sua execução, comutar para outra(s) tarefa(s) e mais tarde retomar a execução da primeira tarefa a partir do ponto onde foi interrompida.

O contexto de uma tarefa é composto por:

- Conteúdo do registo *Program Counter - PC* (par de registos *CS:IP* na arquitectura *Intel x86*) ou registo *PC* dos processadores *ARPA-MT* e *MicroBlaze*. De referir que a arquitectura do processador *PowerPC 405* não disponibiliza nenhum registo para acesso do *Program Counter*;
- Conteúdo do registo de estado, se aplicável (*FLAGS* na arquitectura *Intel x86* e registo *Machine Status Register - MSR* nos processadores *MicroBlaze* e *PowerPC 405*);
- Conteúdo de todos os registos internos do processador utilizáveis pelas tarefas (registos de uso geral, registos de co-processadores, etc.);
- Pilha (*stack*) e respectivo ponteiro.

Além das tarefas, o sistema é constituído por *N* semáforos. O primeiro (índice 0) é dedicado ao controlo de acesso aos co-processadores *Cop0-MEC* e *Cop2-OSC* do processador *ARPA-MT*. Os restantes *N-1* semáforos estão disponíveis para fins genéricos dependentes da

aplicação. No entanto, ao contrário do que acontece com as tarefas, durante a configuração do executivo, o número máximo de semáforos pode ser especificado como sendo zero.

Nas implementações em *software*, o índice 0 quer das tarefas, quer dos semáforos é usado para fins específicos, sendo que o valor zero é também usado com o significado de identificador inválido. No entanto, para a versão OReK-CP, o índice 0 das tarefas e dos semáforos, é perfeitamente usável sendo o código de erro diferente para esta implementação.

A.3.1 Núcleo

O componente central do executivo *OReK* é o seu núcleo. Internamente, o núcleo responde às interrupções do temporizador e dos periféricos, executa o algoritmo de escalonamento e efectua a comutação das tarefas. Disponibiliza à aplicação os seguintes serviços:

- Inicialização e terminação do núcleo;
- Criação e destruição de tarefas;
- Arranque e paragem de tarefas;
- Informação temporal e de estado das tarefas;
- Gestão de grupos de tarefas;
- Criação, destruição e gestão de semáforos;
- Controlo da preempção das tarefas.

O temporizador de *hardware* gera interrupções periodicamente. Sempre que ocorre uma interrupção é executado o algoritmo de escalonamento das tarefas e eventualmente feita a comutação de tarefas. Tudo isto é feito no âmbito da rotina de serviço à interrupção do temporizador. No início desta rotina é necessário salvaguardar o contexto da tarefa actual, seguidamente actualizar os parâmetros temporais dinâmicos das tarefas, determinar a próxima tarefa a executar e no final da rotina restaurar o contexto da próxima tarefa que vai ser executada.

No caso de no contexto da aplicação ser executada uma operação que faça com que a tarefa activa transite para um estado não activo (e.g. terminação da tarefa) é também gerada por *software* uma pseudo-interrupção do temporizador de forma a que seja invocada a respectiva rotina de serviço onde é feito o escalonamento e a comutação de tarefas.

O conteúdo do *PC* é automaticamente salvaguardado quando ocorre uma interrupção. O registo de estado é automaticamente salvaguardado por *hardware* na arquitectura *Intel x86*. O mesmo não acontece nos processadores *MicroBlaze* e *PowerPC 405* onde a salvaguarda do registo de estado tem de ser feita explicitamente por *software*. Os restantes registos também têm de ser explicitamente salvaguardados pelo executivo.

A troca da pilha activa é feita através da alteração do valor do registo usado como ponteiro da pilha de forma a apontar para o topo da pilha da próxima tarefa que vai ser executada.

A.3.2 Tarefas

As tarefas são objectos que encapsulam o código e os dados e implementam a funcionalidade da aplicação. No executivo *OReK*, uma tarefa pode estar num dos seguintes estados:

- *Free* - (Livre);
- *Stopped* - (Parada);
- *Idle* - (Inactiva);
- *Ready* - (Pronta);
- *Running* - (A executar);
- *Preempted* - (Interrompida).

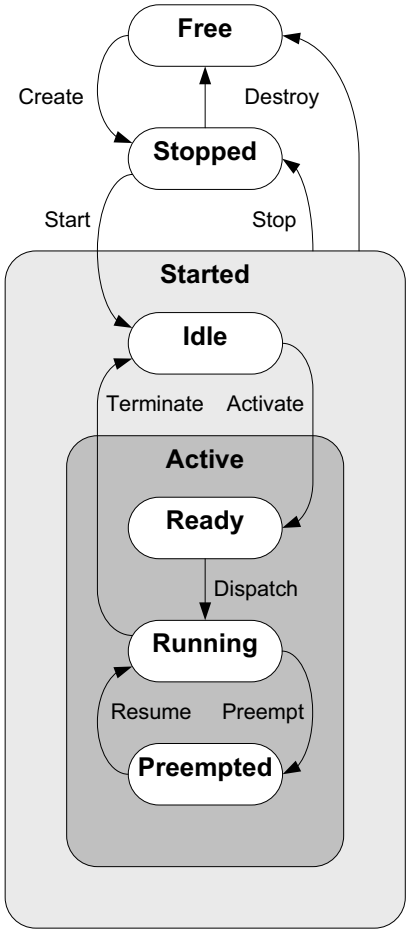


Figura A.2: Estados das tarefas e possíveis transições no executivo *OReK* (retirado de [ASRdO07]).

A figura A.2 ilustra os vários estados em que uma tarefa se pode encontrar, bem como as transições permitidas.

Qualquer tarefa que se encontre num dos estados *Idle*, *Ready*, *Running* ou *Preempted* diz-se iniciada (*Started*), caso contrário diz-se parada (*Stopped*). Uma tarefa que se encontre num dos estados *Ready*, *Running* ou *Preempted* diz-se activa (*Active*).

Uma tarefa quando é criada é colocada no estado *Stopped*, permanecendo nesse estado até ser explicitamente iniciada. Depois de iniciada é colocada no estado *Idle* até ser alcançado o instante da primeira activação. Nessa altura transita para o estado *Ready*. Uma tarefa no estado *Ready* está pronta a executar. A passagem para o estado *Running* é feita pelo algoritmo de escalonamento através da atribuição de tempo de processador à tarefa. Uma tarefa a executar pode ser interrompida de forma a ser executada outra mais prioritária, passando nesta situação para o estado *Preempted*. Uma tarefa depois de terminar é destruída ou colocada no estado *Idle* até à próxima activação ou instância.

Em qualquer estado pode ser dada ordem de paragem ou de destruição de uma tarefa, transitando neste caso para o estado *Stopped* ou *Free*, respectivamente. A transição de estado pode ser deferida ou imediata, consoante a tarefa já tenha iniciado ou não a sua execução, respectivamente. Uma tarefa depois de destruída, deixa de existir no sistema.

Os estados *Sleeping*, *Suspended* e *Blocked* normalmente implementados nos executivos ou sistemas operativos de tempo-real, não foram incluídos no executivo *OReK* pelos motivos expostos no capítulo 2.5, relacionados com a implementação do protocolo *SRP*.

A.3.3 Semáforos

Os semáforos são objectos que encapsulam os mecanismos de sincronização do executivo *OReK*. Um semáforo pode estar num dos seguintes estados:

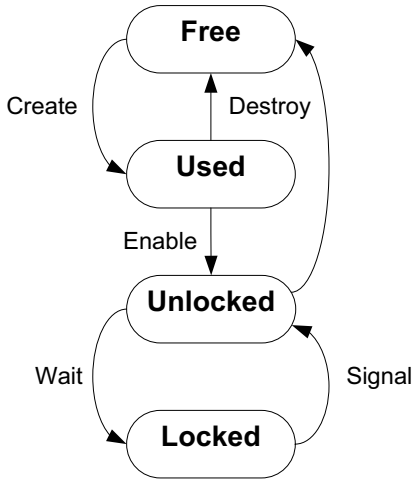


Figura A.3: Estados dos semáforos e possíveis transições no executivo *OReK* (retirado de [ASRdO07]).

- *Free* - (Livre);
- *Used* - (Usado);
- *Unlocked* - (Desbloqueado);

- *Locked* - (Bloqueado).

A figura A.3 ilustra os vários estados em que um semáforo se pode encontrar, bem como as transições permitidas.

Um semáforo quando é criado é colocado no estado *Used*. Neste estado deve ser inicializado o respectivo tecto do recurso. Seguidamente pode ser activado, transitando para o estado *Unlocked*. Um semáforo transita do estado *Unlocked* para o estado *Locked* através da execução da primitiva *Wait*. O retorno ao estado *Unlocked* faz-se por intermédio da primitiva *Signal*. Um semáforo pode ser destruído de não estiver no estado *Locked*. Um semáforo depois de destruído, deixa de existir no sistema.

A.4 Implementação

As próximas subsecções descrevem alguns detalhes relativos à implementação do executivo *OReK*, nomeadamente, as linguagens de programação, as classes e estruturas, a organização dos ficheiros e a realização da gestão de tarefas e da sua sincronização com base em semáforos.

A.4.1 Linguagens e Independência da Plataforma

A generalidade do executivo *OReK* é independente da plataforma alvo e está escrito em *C++* portável. A linguagem *C++* foi a escolhida por ser:

- Eficiente;
- Fácil de realizar a interface com módulos escritos noutras linguagens, nomeadamente em *C* e *assembly*;
- Orientada por objectos;
- Popular e bem suportada por ferramentas para diversas plataformas.

Algumas partes de baixo nível, mas independentes da plataforma estão implementadas em linguagem *C*, igualmente portável.

Existem ainda pequenos segmentos de código que são específicos da plataforma, tais como as funções de programação do temporizador, o atendimento de baixo nível das interrupções dos periféricos, as funções onde se acede explicitamente aos registos do processador e os segmentos da rotina de serviço à interrupção do temporizador onde é feita a comutação do contexto da tarefa. Nestes casos foi utilizada a linguagem *assembly* do processador alvo.

Apesar de no estado actual o executivo *OReK* ser suportado apenas nas plataformas *Intel x86/MSDOS*, *ARPA-MT*, *MB-SoC* e *PPC-SoC*, houve o cuidado de isolar todos os blocos de código que sejam dependentes da plataforma. Desta forma é relativamente simples portar o executivo para outras plataformas, bastando para tal reescrever esses blocos.

Um aspecto específico da arquitectura é a pilha da tarefa. De forma a poder ser feita a comutação das tarefas, sempre que ocorre uma interrupção do temporizador, o contexto da tarefa interrompida deve ser armazenado na pilha da tarefa, sendo o respectivo ponteiro

guardado no bloco de controlo dessa tarefa. Essa informação será usada posteriormente quando for retomada a execução da tarefa interrompida.

A.4.2 Classes e Estruturas

A figura A.4 ilustra as componentes, isto é, os módulos e as classes principais do executivo *OReK* que implementam a arquitectura representada na figura A.1.

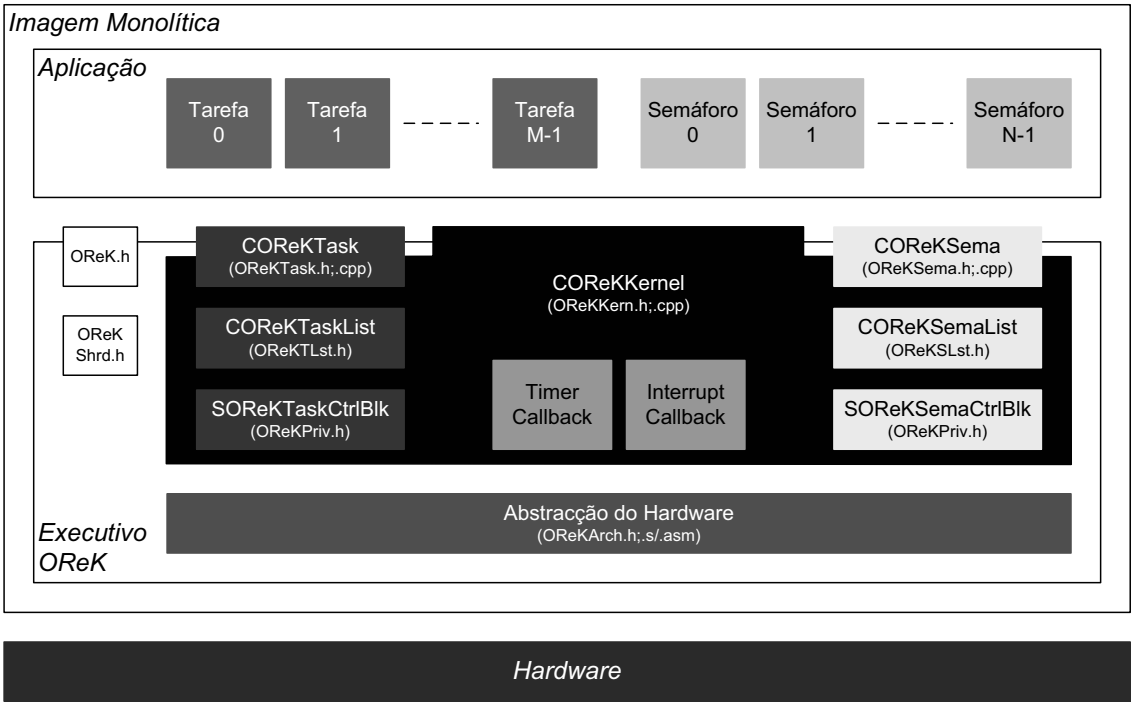


Figura A.4: Estrutura da implementação em software do executivo *OReK* (retirado de [ASRdO07]).

Mais concretamente são mostradas as classes `COReKKern`, `COReKTask`, `COReKTaskList`, `COReKSema` e `COReKSemaList` e as estruturas de dados `SOReKTaskCtrlBlk` e `SOReKSemaCtrlBlk`. São também mostradas na figura A.4 as funções executadas no contexto das rotinas de serviço às interrupções do temporizador e dos periféricos (*timer callback* e *interrupt callback*). As subsecções seguintes descrevem sucintamente cada um destes elementos.

A.4.2.1 A Classe COReKKern

A classe `COReKKern` implementa o núcleo do executivo *OReK*. As figuras A.5 e A.6 mostram, respectivamente, a interface (métodos públicos) e os atributos protegidos desta classe. Em conjunto implementam os serviços disponibilizados pelo núcleo e enumerados acima.

Os nomes dos métodos e atributos são auto-explicativos pelo que não serão considerados individualmente para não tornar esta discussão enfadonha. Além disso, os métodos relacionados com a gestão de tarefas e de semáforos não se destinam a ser invocados directamente

```
class COReKKern{
public:  // Methods
    static int Initialize(uint maxNumTasks, uint maxNumSemas, uint baseFrequency, uint timeResolution
    #ifndef __ARPA__
        #if !defined (__MICROBLAZE__) && !defined (__PPC__)
                        , ostream* pLogStream = NULL
        #endif // __MICROBLAZE__  && __PPC__
    #endif // __ARPA__
                         );
    static int ShutDown(void);
    static uhuge GetTickCount(void);
    static void DisablePreemption(void);
    static void EnablePreemption(void);

    static int CreateNonRTTask(COReKTask* pTask, handle* phTask, uchar basePrio, bool singlShot,
                               uchar taskGrpId = 0,
                               uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                               uchar runContxt = 0);

    static int CreateSoftRTPerTask(COReKTask* pTask, handle* phTask, int period, int initPhase,
                                   uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int CreateSoftRTApeTask(COReKTask* pTask, handle* phTask, int minITime, int intBind,
                                   uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int CreateHardRTPerTask(COReKTask* pTask, handle* phTask, int period, int relDdlin,
                                   int initPhase, uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int CreateHardRTApeTask(COReKTask* pTask, handle* phTask, int minITime, int relDdlin,
                                   int intBind, uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int DestroyTask(handle hTask);
    static int StartTask(handle hTask, int initPhase);
    static int StopTask(handle hTask);
    static int StartTaskGroup(uchar taskGrpId);
    static int StopTaskGroup(uchar taskGrpId);
    static int StartAllTasks(void);
    static int StopAllTasks(void);

    static int ActivateTask(handle hTask, int delay);
    static int GetTaskState(handle hTask, TOReKTaskState* pTaskState);
    static int CreateSema(COReKSema* pSema, handle* phSema);
    static int DestroySema(handle hSema);
    static int RegTaskOnSema(handle hSema, handle hTask, bool enable);
    static int EnableSema(handle hSema);
    static int Wait(handle hSema);
    static int Signal(handle hSema);

    static const char* GetStatusTypeStr(int statusCode);
    static const char* GetStatusMsgStr(int statusCode);
};
```

Figura A.5: Interface da classe `COReKKern` do executivo *OReK*.

```
// Attributes
protected:
static bool m_running;
static uint m_baseFrequency;
static uint m_timeResolution;
static uint m_kernAuxStacks[OREK_NUM_CONTXTS][OREK_KERN_AUX_STACK_SZ];

#ifndef  __ARPA_OSC__

    static uhuge m_tickCount;

    static bool m_preemptEnabled[OREK_NUM_CONTXTS];

    static bool m_taskEndException[OREK_NUM_CONTXTS];
    static bool m_signalException[OREK_NUM_CONTXTS];

#endif // __ARPA_OSC__

    static int m_maxNumTasks;
    static TOReKTaskCtrlBlk* m_pTCBPool;

#ifndef  __ARPA_OSC__

    static COReKTaskList m_freeTCBs;

    static COReKTaskList m_stoppedTaskList;
    static COReKTaskList m_idlePerTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_idleApeTaskList;
    static COReKTaskList m_readyNonRTTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_readySoftRTTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_readyHardRTTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_preemptedTaskList[OREK_NUM_CONTXTS];

    static TOReKTaskCtrlBlk* m_pIntBindTasks[OREK_NUM_INTERRUPTS];

#endif // __ARPA_OSC__

    static TOReKTaskCtrlBlk* m_pRunningTasks[OREK_NUM_CONTXTS];

    static int m_maxNumSemas;
    static TOReKSemaCtrlBlk* m_pSCBPool;

#ifndef  __ARPA_OSC__

    static COReKSemaList m_freeSCBs;

    static uint* m_pCeilStackBlks;
    static uint* m_pPrevContxtsCeil[OREK_NUM_CONTXTS];
    static uint m_contxtsCeiling[OREK_NUM_CONTXTS];

#endif // __ARPA_OSC__

#ifndef __ARPA__
    #if !defined (__MICROBLAZE__) && !defined (__PPC__)
        static ostream* m_pLogStream;
    #endif // __MICROBLAZE__ && __PPC__
#endif // __ARPA__
```

Figura A.6: Atributos da classe `COReKKern` do executivo *OReK*.

pela aplicação mas indirectamente através das respectivas classes `COReKTask` e `COReKSema`.

Todos os métodos e atributos desta classe são estáticos e a classe não é instanciável porque não faz sentido existir mais do que um núcleo no sistema.

Os atributos listados na figura A.6 entre as directivas `#ifndef __ARPA_OSC__` e `#endif // __ARPA_OSC__` são incluídos nas implementações *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC* uma vez que na versão *OReK-ARPA-C2* correspondem a campos dos registos do co-processador *Cop2-OSC*.

A.4.2.2 A Classe COReKTask

A classe `COReKTask` fornece uma definição abstracta de uma tarefa, pelo que não pode ser directamente instanciada. Esta classe simplifica a manipulação das tarefas, guardando internamente o identificador ou *handle* da tarefa devolvido pelo núcleo aquando da criação da mesma. Sempre que necessário utiliza esse identificador para invocar a função do núcleo correspondente. A figura A.7 mostra a interface desta classe e que consiste num construtor, num destrutor e nos seguintes métodos:

- `CreateNonRT` - cria uma tarefa ordinária;
- `CreateSoftRTPer` - cria uma tarefa periódica;
- `CreateSoftRTApe` - cria uma tarefa aperiódica;
- `CreateHardRTPer` - cria uma tarefa periódica de tempo-real;
- `CreateHardRTApe` - cria uma tarefa aperiódica de tempo-real;
- `Destroy` - destrói a tarefa;
- `Start` - arranca a tarefa;
- `Stop` - pára a tarefa;
- `Activate` - activa a tarefa (aplicável apenas a tarefas aperiódicas);
- `GetState` - devolve o estado da tarefa.

Além destes métodos, define ainda a assinatura do método de entrada da tarefa (`virtual void Main() = 0`). Este método não está implementado nesta classe e deve ser definido em todas as classes que implementam tarefas concretas, as quais têm que ser derivadas da classe `COReKTask`.

Esta classe pode invocar os serviços do núcleo do executivo directamente ou através de chamadas ao sistema. A primeira abordagem pode ser empregue quando a aplicação e o executivo são integrados numa única imagem monolítica. A segunda abordagem é usada quando a aplicação e o executivo são carregados como módulos separados ou executam em diferentes modos de operação do processador ou espaços de endereçamento.

A.4.2.3 A Classe COReKSema

A classe `COReKSema` define as funcionalidades acessíveis à aplicação disponibilizadas pelos semáforos implementados no executivo *OReK*. Esta classe simplifica a manipulação dos semáforos, guardando internamente o identificador ou *handle* do semáforo devolvido pelo núcleo aquando da criação do mesmo. Sempre que necessário utiliza esse identificador para invocar a função do núcleo correspondente.

```
class COReKTask
{
    friend class COReKSema;

// Constructors / Destructors prototypes
public:
    COReKTask();
    virtual ~COReKTask();

// Methods prototypes
public:
    int CreateNonRT(uchar basePrio, bool singlShot,
                    uchar taskGrpId = 0,
                    uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                    uchar runContxt = 0);

    int CreateSoftRTPer(int period, int initPhase,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int CreateSoftRTApe(int minITime, int intBind,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int CreateHardRTPer(int period, int relDdlin, int initPhase,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int CreateHardRTApe(int minITime, int relDdlin, int intBind,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int Destroy();
    int Start(int initPhase);
    int Stop();
    int Activate(int delay);

    int GetState(TOReKTaskState* pTaskState);

    virtual void Main() = 0;

// Attributes
protected:
    handle m_handle;
};
```

Figura A.7: Interface da classe `COReKTask` do executivo *OReK*.

Ao contrário do que acontecia com as tarefas, esta classe pode ser instanciada directamente. A figura A.8 ilustra a interface pública desta classe, sendo constituída por um constructor, um destructor e pelos seguintes métodos:

- `Create` - cria um semáforo;
- `Destroy` - destrói o semáforo;
- `Enable` - activa o semáforo;
- `RegisterTask` - regista uma tarefa como utilizadora do recurso controlado pelo semáforo;
- `Wait` - bloqueia o semáforo;
- `Signal` - desbloqueia o semáforo.

```
class COReKSema
{
    friend class COReKTask;

// Constructors / Destructors prototypes
public:
    COReKSema();
    virtual ~COReKSema();

// Methods prototypes
public:
    int Create();
    int Destroy();

    int RegisterTask(const COReKTask& task, bool enable = false);
    int Enable();

    int Wait();
    int Signal();

// Attributes
protected:
    handle m_handle;
};
```

Figura A.8: Interface da classe `COReKSema` do executivo *OReK*.

Após a construção de um semáforo, todas as tarefas que acedem ao recurso controlado pelo semáforo, devem ser registadas de forma a configurar o respectivo tecto. Este requisito deve-se ao facto de, apesar das suas importantes vantagens, a utilização do protocolo *SRP* na gestão dos semáforos não ser transparente para a aplicação.

Tal como na classe `COReKTask`, esta classe pode invocar os serviços do núcleo do executivo directamente ou através de chamadas ao sistema.

A.4.2.4 A Estrutura SOReKTaskCtrlBlk

A estrutura `SOReKTaskCtrlBlk` define o bloco de controlo da tarefa (*Task Control Block - TCB*). Esta estrutura possui vários campos onde são armazenados os parâmetros de configuração e guardado o estado da respectiva tarefa. A sua definição possui duas partes (figura A.9):

- A primeira (antes da directiva `#ifndef __ARPA_OSC__`) - independente da implementação;

- A segunda (depois da directiva `#ifndef __ARPA_OSC__`) - incluída nas versões *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC*, uma vez que estes campos fazem parte do co-processador *Cop2-OSC* na implementação *OReK-ARPA-C2*.
- A terceira (ver figura A.10), inclui a versão para a implementação do sistema OReK-CP, também devido à maioria dos campos estarem implementados no co-processador genérico.

O campo `m_pTask` é um ponteiro para um objecto ou instância de uma classe derivada da `COReKTask` onde é implementada a funcionalidade da tarefa propriamente dita. É através deste ponteiro que o executivo invoca o método de entrada da tarefa.

O campo `m_stackSize` contém o tamanho da pilha associada à tarefa medido em palavras nativas da arquitectura base (palavra de 16 bits na plataforma *Intel x86/MSDOS* ou de 32 bits na plataforma *ARPA-MT* e nos processadores *MicroBlaze* e *PowerPC 405*). Este valor é estabelecido durante a criação da tarefa não podendo ser alterado posteriormente.

O campo `m_pStackBlk` armazena o endereço inicial do bloco de memória onde reside a pilha da tarefa. O campo `m_currentSP` armazena o valor do registo da *CPU* que desempenha o papel de *stack pointer* inicial da tarefa ou no instante em que é interrompida, para que possa ser retomada mais tarde. De notar que todo o estado da tarefa é armazenado na respectiva pilha, pelo que é suficiente guardar o endereço do topo no momento da preempção.

Nas implementações *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC*, uma tarefa está numa das listas de tarefas internas do núcleo. Cada lista possui um estado associado. O campo `m_pTaskList` é um ponteiro para a lista na qual a tarefa se encontra num dado momento. Os campos `m_pPrevious` e `m_pNext` são ponteiros utilizados pela classe `COReKTaskList` na implementação das listas bi-ligadas de *TCBs*.

Para optimizar simultaneamente o acesso arbitrário a qualquer bloco assim como a inserção ou remoção de blocos, as listas de tarefas são bi-ligadas e implementadas sobre uma tabela (*array*) de blocos de tamanho predefinido alocada estaticamente. Algumas das listas são mantidas ordenadas enquanto outras são manipuladas como filas. Sobre a mesma tabela são implementadas várias listas correspondendo cada uma a um dos estados em que uma tarefa se pode encontrar. Além dessas, existe também uma lista de blocos livres. A utilização de uma tabela tem a vantagem de simplificar a validação do identificador (*handle*) da tarefa, o qual corresponde ao índice do *TCB* dentro da tabela.

A utilização de uma tabela estática de tamanho predefinido tem também a vantagem de eliminar o tempo de alocação/libertação de memória necessário numa solução dinâmica, obviamente à custa de uma diminuição da flexibilidade. No entanto, é importante notar que é sempre possível realizar a realocação e redimensionamento da tabela de *TCBs*, embora essa seja uma operação que obriga a suspender temporariamente a execução das tarefas.

O atributo `m_headerWord`/`m_headerBits` é uma *union* que implementa em *software* um registo com funcionalidade análoga ao campo *THEADER* do *TCB* implementado no co-processador *Cop2-OSC*, armazenando informação sobre o tipo da tarefa, o seu estado actual, o contexto onde executa, o grupo a que pertence, a associação a fontes de interrupção (possível

```
typedef struct SOReKTaskCtrlBlk
{
    COReKTask* m_pTask;

    uint  m_stackSize;
    uint* m_pStackBlk;

    uint  m_currentSP;

#ifndef  __ARPA_OSC__

    COReKTaskList*    m_pTaskList;
    SOReKTaskCtrlBlk* m_pPrevious;
    SOReKTaskCtrlBlk* m_pNext;

    union
    {
        struct
        {
            uint m_taskType:3;
            uint m_dtryPend:1;
            uint m_stopPend:1;
            uint m_actvPend:1;
            uint m_rsvdBits1:3;
            uint m_ddlinMiss:1;
            uint m_rsvdBits2:2;
            uint m_runContxt:3;
            uint m_singlShot:1;
            uint m_taskGrpId:8;
            uint m_intRqBind:1;
            uint m_intNumber:4;
            uint m_rsvdBits3:3;
        }m_headerBits;
        uint m_headerWord;
    };

    union
    {
        uint m_basePrio;
        uint m_period;
        uint m_minITime;
    };

    union
    {
        uint m_preptLev;
        uint m_relDdlin;
    };

    uint m_activCnt;

    uint m_priority;

#endif // __ARPA_OSC__

}TOReKTaskCtrlBlk;
```

Figura A.9: Definição da estrutura `SOReKTaskCtrlBlk` do executivo *OReK*.

```
typedef struct SOReKTaskCtrlBlk{
int id;

   COReKTask* m_pTask;
   uint  m_stackSize;
   uint* m_pStackBlk;
   uint  m_currentSP;

   union{
      struct{
        uint m_taskType:3;
        uint m_dtryPend:1;
        uint m_stopPend:1;
        uint resvbits:3;
        uint m_taskGrpId:8;
      }m_headerBits;
      uint short m_headerWord;
   };
}TOReKTaskCtrlBlk;
```

Figura A.10: Definição da estrutura `SOReKTaskCtrlBlk` do executivo *OReK* para a versão *OReK-CP*.

nas tarefas aperiódicas), além de outros indicadores.

O atributo `m_basePrio/m_period/m_minITime` é uma *union* que implementa em *software* um registo com funcionalidade análoga ao campo *TPERIOD* do *TCB* implementado no co-processador *Cop2-OSC*, armazenando a prioridade base, o período ou o intervalo mínimo entre activações consoante se trate de uma tarefa ordinária, periódica ou aperiódica, respectivamente.

O atributo `m_preptLev/m_relDdlin` é uma *union* que implementa em *software* um registo com funcionalidade análoga ao campo *TPREPLV* do *TCB* implementado no co-processador *Cop2-OSC*, armazenando o nível de preempção ou a *deadline* relativa das tarefas.

O atributo `m_activCnt` implementa em *software* um registo com funcionalidade análoga ao campo *TACTIVC* do *TCB* implementado no co-processador *Cop2-OSC*, armazenando a fase inicial das tarefas periódicas ou o atraso de activação das tarefas aperiódicas. Além disso, é também usado como contador para gerir as reactivações das tarefas periódicas e o intervalo mínimo entre activações das tarefas aperiódicas.

O atributo `m_priority` implementa em *software* um registo com funcionalidade análoga ao campo *TPRIORI* do *TCB* implementado no co-processador *Cop2-OSC*, armazenando a prioridade actual da tarefa.

O atributo `id` presente apenas na implementação *OReK-CP*, representa o índice da tarefa dentro do co-processador.

A.4.2.5 A Classe COReKTaskList

A classe `COReKTaskList` implementa uma lista bi-ligada de estruturas de dados do tipo `SOReKTaskCTRLBlk` sobre uma tabela predefinida de tamanho fixo. Esta classe disponibiliza os serviços tradicionais para inserção e remoção de elementos da lista além de outras funcionalidades úteis para o escalonamento de tarefas. Para melhorar o desempenho, esta classe utiliza as capacidades *inline* do C++.

A.4.2.6 A Estrutura SCOReKSemaCtrlBlk

A estrutura `SOReKSemaCtrlBlk` define o bloco de controlo do semáforo (*Semaphore Control Block - SCB*). Esta estrutura possui vários campos onde são armazenados os parâmetros de configuração e guardado o estado do respectivo semáforo. A sua definição possui duas partes (ver figura A.11):

- A primeira (antes da directiva `#ifndef __ARPA_OSC__`) - independente da implementação.
- A segunda (depois da directiva `#ifndef __ARPA_OSC__`) - incluída nas versões *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC*, uma vez que estes campos fazem parte do co-processador *Cop2-OSC* na implementação *OReK-C2*.
- A terceira (ver figura A.12), inclui a versão para a implementação do sistema OReK-CP, também devido à maioria dos campos estarem implementados no co-processador genérico.

```
typedef struct SOReKSemaCtrlBlk
{
    COReKSema* m_pSema;

#ifndef  __ARPA_OSC__

    COReKSemaList* m_pSemaList;
    SOReKSemaCtrlBlk* m_pNext;

    union
    {
        struct
        {
            uchar m_used:1;
            uchar m_enable:1;
            uchar m_locked:1;
        }m_headerBits;
        uchar m_headerWord;
    };

    uint m_resourceCeil;

#endif // __ARPA_OSC__

} TOReKSemaCtrlBlk;
```

Figura A.11: Definição da estrutura `SOReKSemaCtrlBlk` do executivo *OReK*.

```
typedef struct SOReKSemaCtrlBlk{
   COReKSema* m_pSema;
   int id;
   bool locked;
}TOReKSemaCtrlBlk;
```

Figura A.12: Definição da estrutura `SOReKSemaCtrlBlk` do executivo *OReK* para a implementação OReK-CP.

O campo `m_pSema` é um ponteiro para um objecto ou instância da classe `COReKSema` usada para representar o semáforo do lado da aplicação.

Nas implementações *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC*, os *SCBs* livres são guardados numa lista simplesmente ligada. O campo `m_pSemaList` é um ponteiro para a lista na qual o semáforo se encontra num dado momento. O campo `m_pNext` é um ponteiro utilizado pela classe `COReKSemaList` na implementação das listas de *SCBs*.

Tal como acontecia com as tarefas, nos semáforos para optimizar simultaneamente o acesso arbitrário a qualquer bloco de controlo assim como a inserção ou remoção de blocos, as listas de semáforos estão implementadas sobre uma tabela (*array*) de blocos de tamanho predefinido alocada estaticamente.

Os indicadores `m_used`, `m_enable` e `m_locked` indicam o estado do semáforo, possuindo o significado alocado, activo (desbloqueado) e bloqueado, respectivamente.

O campo `m_resourceCeil` armazena o tecto do recurso controlado pelo semáforo, sendo definido como o maior nível de preempção de todas as tarefas que acedem ao recurso controlado em regime de exclusão mútua.

O campo `id` exclusivo da implementação *OReK-CP*, identifica o índice do semáforo dentro do co-processador.

A.4.2.7 A Classe COReKSemaList

A classe `COReKSemaList` implementa uma lista ligada de estruturas de dados do tipo `SOReKSemaCTRLBlk` sobre uma tabela predefinida de tamanho fixo. Esta classe disponibiliza os serviços tradicionais para inserção e remoção de elementos da lista. Para melhorar o desempenho, esta classe utiliza as capacidades *inline* do *C++*.

A.4.3 Outros Tipos e Constantes do Executivo OReK

No sentido de uniformizar as definições do executivo *OReK*, foram desenvolvidos os ficheiros `OReKShrd.h` e `OReKPriv.h` que contêm as definições partilhadas e privadas do executivo *OReK*, respectivamente.

A.4.3.1 Definições Públicas ou Partilhadas com a Aplicação

```
// Parameters
#define OREK_NUM_CONTXTS             1
#define OREK_NUM_INTERRUPTS          16

#if defined (__MICROBLAZE__) || defined (__PPC__)

    #include <xbasic_types.h>

    #define uchar                    Xuint8
    #define ushort                   Xuint16
    #define ulong                    Xuint32

    typedef unsigned int             uint;
    typedef long long                huge;
    typedef unsigned long long       uhuge;

#else

    typedef unsigned char            uchar;
    typedef unsigned short           ushort;
    typedef unsigned long            ulong;
    typedef unsigned int             uint;

    #ifdef _MSC_VER

        typedef _int64               huge;
        typedef unsigned _int64      uhuge;

    #else

        typedef long long            huge;
        typedef unsigned long long   uhuge;

    #endif // _WIN32
#endif // __MICROBLAZE__ || __PPC__
```

Figura A.13: Parâmetros e definições de tipos de dados do executivo *OReK*.

No ficheiro de definições partilhadas do executivo *OReK* (`OReKShrd.h`) constam como parâmetros o número de contextos e número de interrupções do executivo, são definidos nomes concretos para os tipos de dados utilizados no executivo *OReK*, de forma a que todas as definições de tipos de dados no executivo sejam independentes da plataforma. São ainda definidos os códigos de estado do executivo *OReK* e definidas as enumerações `EOReKTaskType`, `EOReKTaskState` e `EOReKSemaState`. Estas definem os tipos e estados das tarefas e estados permitidos para os semáforos, respectivamente. As figuras A.13, A.14 e A.16 ilustram as definições partilhadas do executivo *OReK*, sendo que para a implementação *OReK-CP* são adicionadas as definições ilustradas na figura A.15.

```
#define OREK_INVALID_HANDLE_VALUE               -1

#define OREK_INFO_SUCCESS                       0x00

#define OREK_WARNING_TASK_DESTROY_PENDING       0x01
#define OREK_WARNING_TASK_STOP_PENDING          0x02

#define OREK_ERROR_UNKNOWN_ERROR                0x10
#define OREK_ERROR_OUT_OF_MEMORY                0x11
#define OREK_ERROR_INVALID_PARAMETER            0x12
#define OREK_ERROR_INVALID_HANDLE               0x13

#define OREK_ERROR_KERNEL_ALREADY_RUNNING       0x18
#define OREK_ERROR_KERNEL_NOT_INITIALIZED       0x19
#define OREK_ERROR_CANNOT_SHUT_DOWN_KERNEL      0x1A
#define OREK_ERROR_DEADLINE_MISSED              0x1B

#define OREK_ERROR_TOO_MANY_TASKS               0x20
#define OREK_ERROR_CANNOT_CREATE_TASK           0x21
#define OREK_ERROR_CANNOT_DESTROY_TASK          0x22
#define OREK_ERROR_CANNOT_STOP_TASK             0x23
#define OREK_ERROR_TASK_ALREADY_CREATED         0x26
#define OREK_ERROR_TASK_ALREADY_STARTED         0x27
#define OREK_ERROR_TASK_ALREADY_STOPPED         0x28
#define OREK_ERROR_TASK_NOT_IDLE                0x29

#define OREK_ERROR_SEMAS_NOT_AVAILABLE          0x30
#define OREK_ERROR_TOO_MANY_SEMAS               0x31
#define OREK_ERROR_CANNOT_CREATE_SEMA           0x32
#define OREK_ERROR_CANNOT_DESTROY_SEMA          0x33
#define OREK_ERROR_SEMA_ALREADY_CREATED         0x34
#define OREK_ERROR_SEMA_ALREADY_LOCKED          0x35
#define OREK_ERROR_SEMA_ALREADY_UNLOCKED        0x36
#define OREK_ERROR_SEMA_NOT_ENABLED             0x37

#define OREK_TASK_DEFAULT_STACK_SZ              256u
#define OREK_NRT_PRIO_IDLE                      0x7
#define OREK_NRT_PRIO_LOW                       0x6
#define OREK_NRT_PRIO_BELOW_NORMAL              0x5
#define OREK_NRT_PRIO_NORMAL                    0x4
#define OREK_NRT_PRIO_ABOVE_NORMAL              0x3
#define OREK_NRT_PRIO_HIGH                      0x2
#define OREK_NRT_PRIO_NEAR_RT                   0x1
```

Figura A.14: Definição dos códigos de estado do executivo *OReK*.

```
#define OREK_CP_TASK_POSITION_ERROR       0x38
#define OREK_CP_INTERRUPT_POSITION_ERROR  0x39
#define OREK_CP_TASK_EMPTY                0x40
#define OREK_CP_ALGORITHM_ERROR           0x41
#define OREK_CP_SEM_POSITION_ERROR        0x42
#define OREK_CP_SEM_STACK_EMPTY           0x43
#define OREK_CP_SEM_STACK_FULL            0x44
#define OREK_CO_TASK_NOT_STOPED           0x45
#define OREK_ERROR_KERNEL_ALREADY_STOPPED 0x46
```

Figura A.15: Definições extras de códigos de estado do executivo *OReK* na implementação OReK-CP.

```
typedef enum EOReKTaskType
{
    TT_UNUSED       = 0,
    TT_NON_RT       = 1,
    TT_SOFT_RT_PER  = 2,
    TT_SOFT_RT_APE  = 3,
    TT_HARD_RT_PER  = 4,
    TT_HARD_RT_APE  = 5
}TOReKTaskType;

typedef enum EOReKTaskState
{
    TS_FREE         = -1,
    TS_STOPPED      = 0,
    TS_IDLE         = 1,
    TS_READY        = 2,
    TS_RUNNING      = 3,
    TS_PREEMPTED    = 4
}TOReKTaskState;

typedef enum EOReKSemaState
{
    SS_FREE         = 0,
    SS_USED         = 1,
    SS_UNLOCKED     = 3,
    SS_LOCKED       = 7
}TOReKSemaState;
```

Figura A.16: Definição dos tipos enumerados `EOReKTaskType`, `EOReKTaskState` e `EOReKSemaState` do executivo *OReK*.

A.4.3.2 Definições Privadas ou Internas ao Executivo

No ficheiro de definições privadas do executivo *OReK* (`OReKPriv.h`) constam essencialmente descrições privadas internas ao executivo, e às estruturas anteriormente descritas `SOReKTaskCtrlBlk` e `SOReKSemaCtrlBlk`. É definido o tamanho mínimo absoluto da pilha das tarefas, tamanho da pilha auxiliar do executivo e valores limite de prioridade para os diversos tipos de tarefas suportadas pelo executivo OReK. A figura A.17 ilustra as definições privadas do executivo *OReK*.

```
// Definitions
#define OREK_TASK_MIN_STACK_SZ              256u
#define OREK_KERN_AUX_STACK_SZ              256u

#define OREK_NON_RT_PRIO_LO_BOUND           OREK_NRT_PRIO_NEAR_RT
#define OREK_NON_RT_PRIO_UP_BOUND           OREK_NRT_PRIO_IDLE

#define OREK_NON_RT_BASE_PRIORITY           0xFFFFFFF8
#define OREK_SOFT_RT_BASE_PRIORITY          0x80000000
```

Figura A.17: Definições privadas do executivo *OReK*.

A.4.4 Funções de Reacção às Interrupções do Temporizador e Periféricos

Nas versões *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC* de implementação exclusiva em *software* são disponibilizadas em linguagem *C* as seguintes duas funções que respondem, do lado do executivo, às interrupções do temporizador e dos periféricos, sendo executadas no contexto das respectivas rotinas de serviço:

- `uint TimerCallback(uchar contxtNum, uint currentSP)` - reage às interrupções do temporizador, guardando o contexto da tarefa actual, actualizando o estado das tarefas, executando o algoritmo de escalonamento e lançando a próxima tarefa mais prioritária em execução;
- `int IntCallback(uchar contxtNum, uchar intSource)` - activa a tarefa aperiódica correspondente à interrupção gerada onde será feito o atendimento propriamente dito.

A.4.5 Escalonamento e Sincronização das Tarefas

A gestão de tarefas é baseada num conjunto de listas internas ao núcleo, nomeadamente:

- *TCBs* livres (`m_freeTCBs`);
- Tarefas paradas (`m_stoppedTaskList`);
- Tarefas periódicas inactivas (`m_idlePerTaskList`);
- Tarefas aperiódicas inactivas (`m_idleApeTaskList`);
- Tarefas ordinárias prontas a executar (`m_readyNonRTTaskList`);
- Tarefas de tempo-real não críticas prontas a executar (`m_readySoftRTTaskList`);
- Tarefas de tempo-real críticas prontas a executar (`m_readyHardRTTaskList`);
- Tarefas interrompidas (`m_preemptedTaskList`).

As listas `m_idlePerTaskList`, `m_readyNonRTTaskList`, `m_readySoftRTTaskList`, `m_readyHardRTTaskList` e `m_preemptedTaskList` possuem uma instância por cada contexto de execução do processador (variável no processador *ARPA-MT* e um nos outros casos). As restantes listas possuem apenas uma instância partilhada por todos os contextos de execução.

As listas `m_idlePerTaskList` e `m_idleApeTaskList` estão ordenadas pelo campo `m_activCnt` de forma a optimizar a activação das tarefas.

As listas de tarefas prontas a executar (`m_readyNonRTTaskList`, `m_readySoftRTTaskList` e `m_readyHardRTTaskList`) estão ordenadas pelo campo `m_priority` de forma a simplificar o escalonamento das tarefas.

A lista `m_preemptedTaskList` funciona como uma pilha de tarefas interrompidas de forma a que as tarefas sejam retomadas por ordem inversa à sua interrupção, simplificando desta forma a implementação do protocolo *SRP*.

As listas `m_freeTCBs` e `m_stoppedTaskList` não possuem qualquer ordenação sendo os *TCBs* colocados em geral no final (cauda) da lista e retirados do início (cabeça) da lista no primeiro caso e de forma arbitrária no segundo caso.

O contador `m_activCnt` dos *TCBs* pertencentes às listas ordenadas (com excepção da `m_readyNonRTTaskList`) é decrementado em todas as unidades de tempo. Quando atingir o valor zero, significa que, no caso das tarefas periódicas foi atingido o instante de uma nova instância e no caso das tarefas aperiódicas que foi cumprido o tempo mínimo entre activações consecutivas, pelo que a partir desse instante a tarefa pode, se necessário, ser activada.

O contador `m_priority` dos *TCBs* pertencentes à lista `m_readyHardRTTaskList` e dos *TCBs* relativos às tarefas de tempo-real críticas na lista `m_preemptedTaskList` é decrementado em todas as unidades de tempo. Quando atingir o valor 0, significa que foi alcançado o tempo limite de execução e se a tarefa ainda não tiver terminado, ocorreu uma violação temporal (*deadline missed*).

O escalonamento baseado em listas ordenadas é bastante simples, uma vez que se baseia unicamente no decremento de contadores e na inserção/remoção de elementos das listas. A preempção de uma tarefa dá-se quando forem cumpridos os requisitos definidos no capítulo 2.5 na discussão do protocolo *SRP*.

No início da operação do executivo *OReK* todas as listas estão vazias excepto a de *TCBs* livres. Esta situação é ilustrada na figura A.18.

Durante a operação do sistema, as tarefas vão mudando de estado, sendo as listas alteradas em conformidade. Na figura A.19 é ilustrada uma possível situação das listas correspondente aos seguintes estados das tarefas (assumindo um sistema com o máximo de 8 tarefas e um processador com um único contexto de execução):

- Tarefa 0 - *Preempted*;
- Tarefa 1 - *Ready*;
- Tarefa 2 - *Idle*;
- Tarefa 3 - *Running*;
- Tarefa 4 - *Stopped*;
- Tarefa 5 - *Idle*;
- Tarefa 6 - *Free*;
- Tarefa 7 - *Free*.

De notar que os *TCBs* representados nas diversas linhas das figuras A.18 e A.19 são os mesmos, isto é, tal como já foi referido acima, as listas são implementadas sobre uma única tabela estática de *TCBs*. Por uma questão de simplicidade apenas são mostradas nas figuras A.18 e A.19 uma lista *idle* e uma lista *ready*, embora na realidade existam várias consoante a criticalidade e o modo de activação das tarefas.

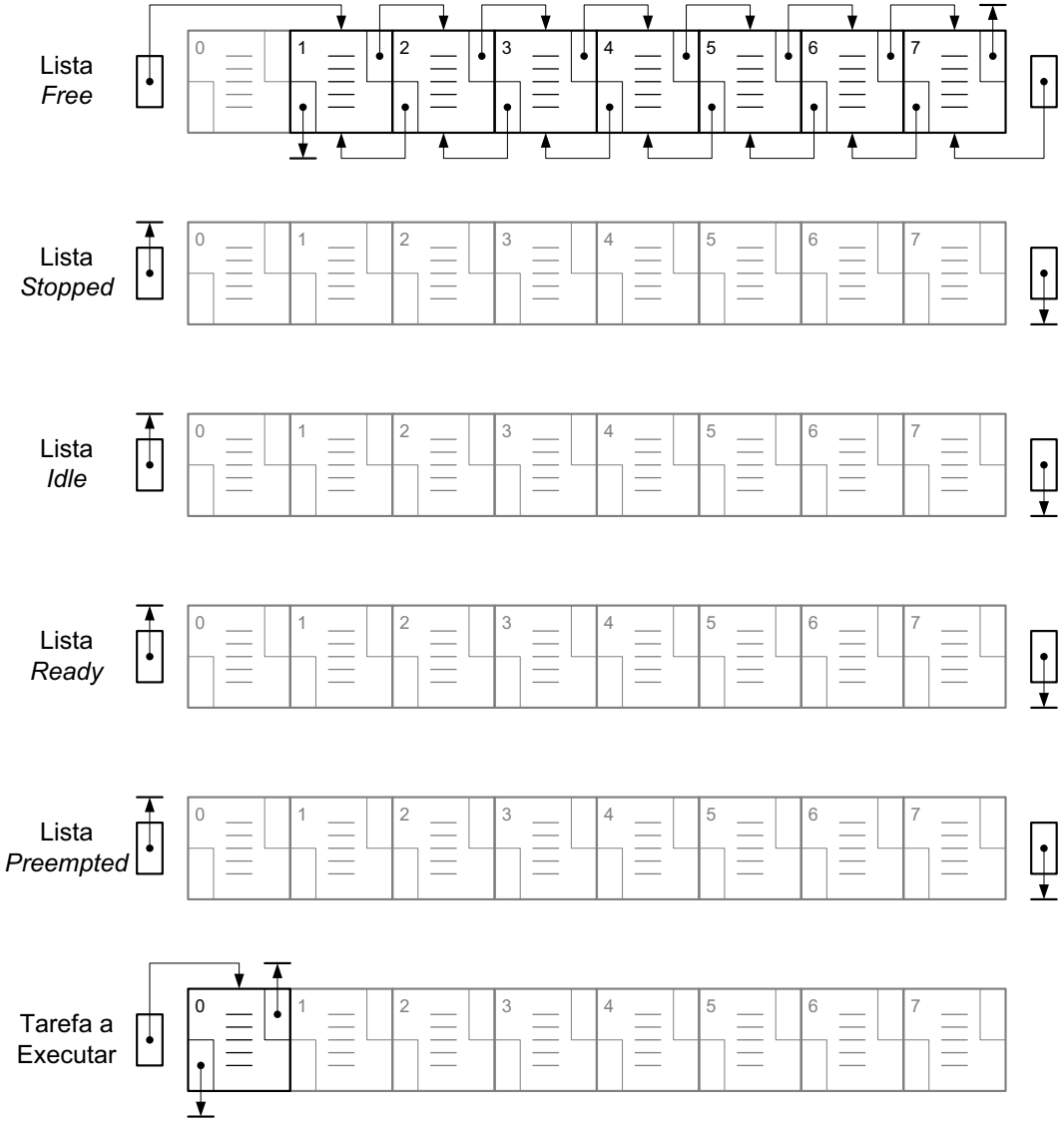


Figura A.18: Estado inicial das listas de *TCBs* (retirado de [ASRdO07]).

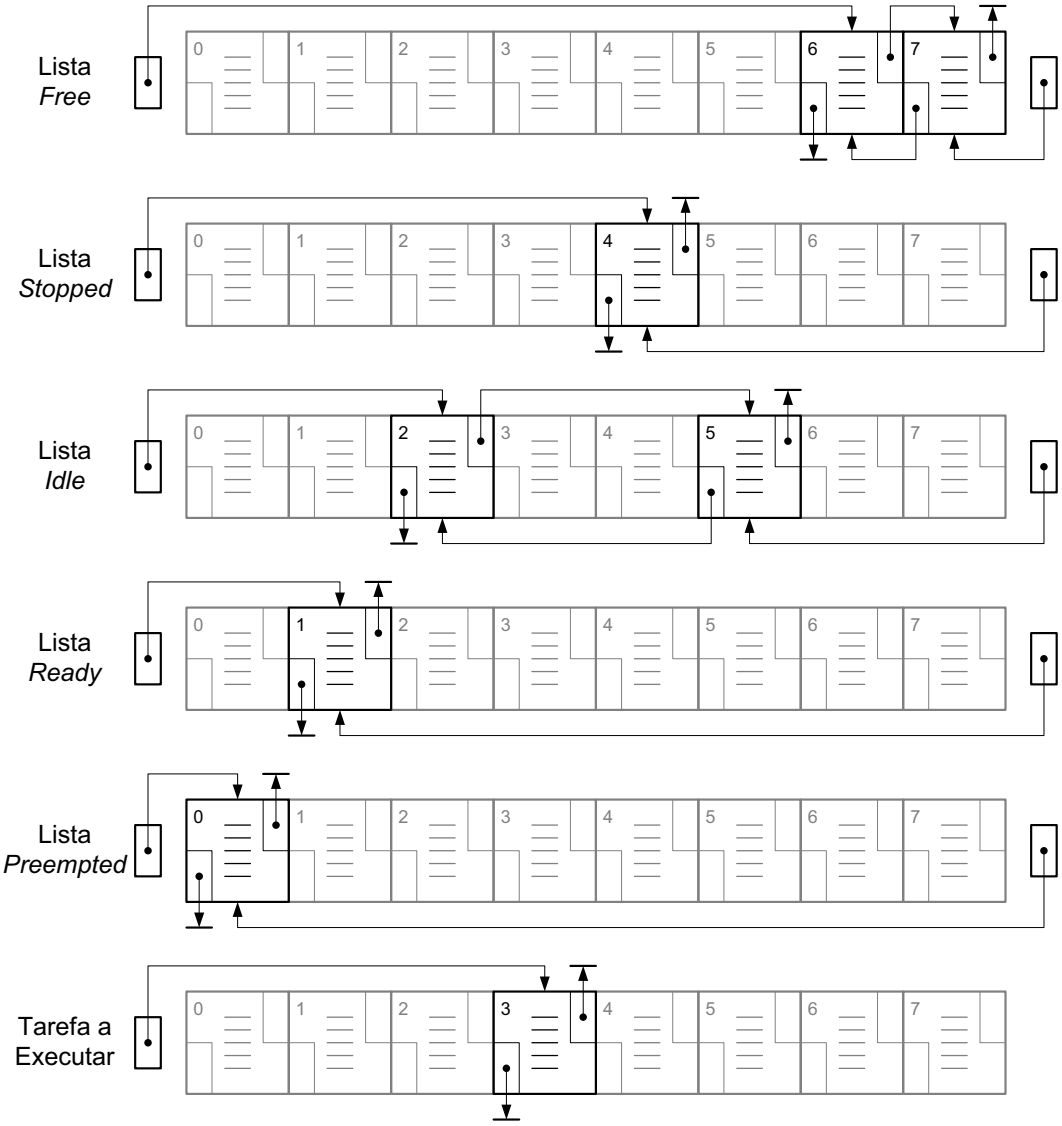


Figura A.19: Possível estado das listas de *TCBs* em pleno funcionamento (retirado de [ASRdO07]).

A.4.6 A Estrutura de Ficheiros

Na figura A.4 é mostrada a relação entre as diversas classes e estruturas de dados que constituem o executivo *OReK* e os ficheiros onde estão implementadas. De uma forma resumida, os ficheiros possuem o seguinte conteúdo:

Ficheiros públicos ou partilhados com a aplicação

- `OReK.h` - ficheiro geral de interface para inclusão nas aplicações (C++/C).
- `OReKShrd.h` - contém definições partilhadas pelo executivo e pela aplicação (C++).
- `OReKKern.h` - contém a definição da classe `COReKKern` (C++).
- `OReKTask.h` - contém a definição da classe `COReKTask` (C++).
- `OReKTask.cpp` - contém a implementação da classe `COReKTask` (C++).
- `OReKTLst.h` - contém a definição e implementação da classe `COReKTaskList` (C++).
- `OReKSema.h` - contém a definição da classe `COReKSema` (C++).
- `OReKSema.cpp` - contém a implementação da classe `COReKSema` (C++).
- `OReKSLst.h` - contém a definição e implementação da classe `COReKSemaList` (C++).
- `MB-PPC.h` - contém a definição da interface do módulo específico das plataformas *MB-SoC* e *PPC-SoC* (C).
- `orek_CoP_Definitions.h` - contém a definição da interface de acesso ao co-processador genérico para a implementação *OReK-CP*.

Ficheiros privados ou internos ao executivo;

- `OReKPriv.h` - contém definições internas (privadas) do executivo (C++).
- `OReKStrs.h` - contém a definição das mensagens intrínsecas do executivo (C).
- `OReKKern.cpp` - contém a implementação da classe `COReKKern` (C++).
- `OReKArch.h` - contém a definição da interface do módulo específico da plataforma (C).
- `OReKARPA.s` - contém a implementação do módulo específico da plataforma *ARPA-MT* (*assembly*).
- `OReKPC.asm` - contém a implementação do módulo específico da plataforma *Intel x86/MSDOS* (*assembly*).
- `OReKXPS.cpp` - contém em linguagem de alto-nível parte da implementação do módulo específico da plataforma *Xilinx* para os processadores *MicroBlaze* e *PowerPC 405* (C).
- `OReKMBLo.s` - contém em linguagem de baixo-nível parte da implementação do módulo específico para o processador *MicroBlaze* (*assembly*).
- `xvectors.S` - contém em linguagem de baixo-nível parte da implementação do módulo específico para o processador *PowerPC 405* (*assembly*).

A.4.7 Versão Suportada pelo co-processador Cop2-OSC

Tal como já foi dito, todas as implementações do executivo *OReK* possuem o mesmo modelo de programação e interface, simplificando desta forma o desenvolvimento e a portabilidade das aplicações.

O diagrama de blocos da figura A.4, relativo à implementação integral em *software* é simplificado no caso da versão *OReK-ARPA-C2* (ver figura A.20), sendo eliminadas as listas de tarefas e de semáforos e as funções invocadas no contexto das interrupções do temporizador e dos periféricos.

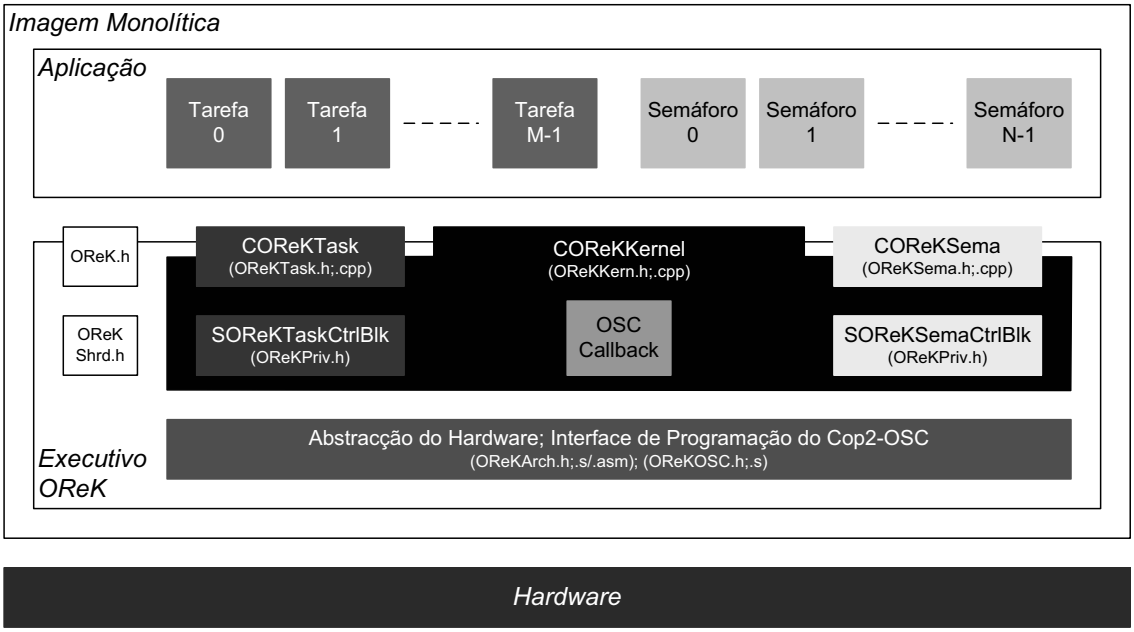


Figura A.20: Estrutura da implementação suportada pelo co-processador *Cop2-OSC* do executivo *OReK* (retirado de [ASRdO07]).

As estruturas de dados são também simplificadas uma vez que parte da informação que elas contêm nas implementações *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC*, passa neste caso a residir no co-processador *Cop2-OSC*. Mais concretamente, os campos delimitados pelas directivas `#ifndef __ARPA_OSC__` e `#endif // __ARPA_OSC__` do pré-processador indicadas nas figuras A.6, A.9 e A.11 não são incluídos nesta versão.

Uma vez que o escalonamento das tarefas, a verificação das condições de preempção e a activação das tarefas aperiódicas para serviço de interrupções são completamente realizadas em *hardware* no co-processador *Cop2-OSC*, nesta implementação existe apenas uma função invocada em caso de excepção para comutação da tarefa em execução. A função usada para este efeito possui o seguinte protótipo e é invocada no contexto da rotina de tratamento das excepções do co-processador *Cop2-OSC*:

```
uint OSCCallback(uchar contxtNum, uint currentSP);
```

O módulo `OReKOSC` representado na figura A.20, o qual contém a interface de programação do *Cop2-OSC* está escrito em *assembly* para que possa tirar partido das instruções especializadas e aceder aos registos disponíveis neste co-processador.

A.4.8 Versão Suportada pelo co-processador OReK-CP

O diagrama de blocos da figura A.4, relativo à implementação integral em *software* é novamente simplificado no caso da versão *OReK-C* (ver figura A.21) tal como foi na versão explicada anteriormente, sendo também eliminadas as listas de tarefas e de semáforos e as funções invocadas no contexto das interrupções do temporizador e dos periféricos.

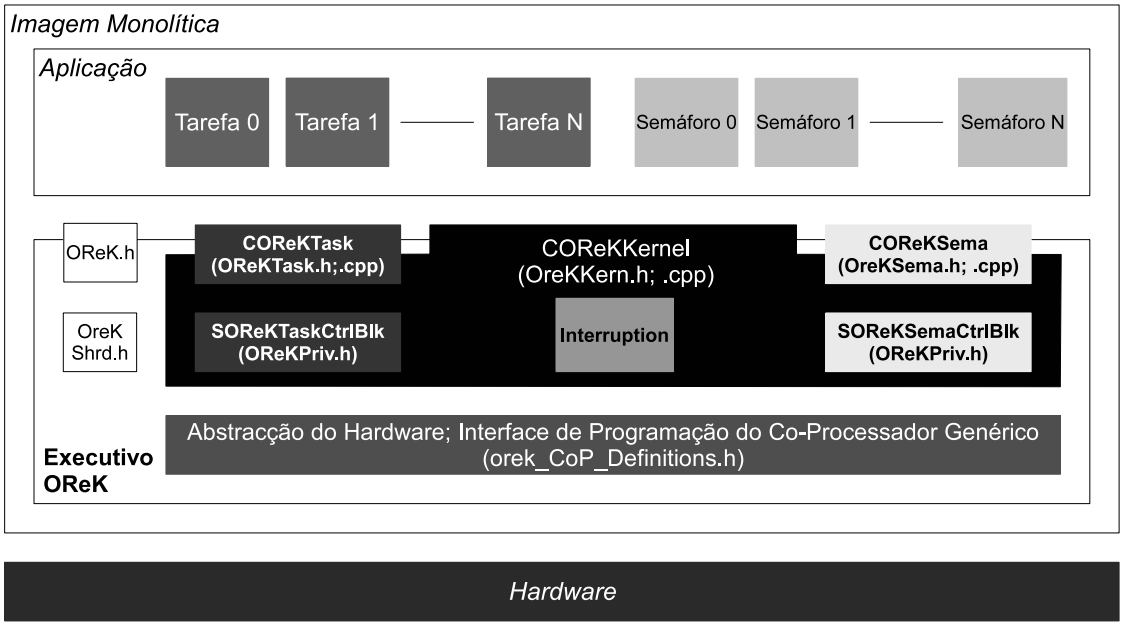


Figura A.21: Estrutura da implementação OReK-CP do executivo *OReK*, utilizando o co-processador genérico

As estruturas de dados foram novamente simplificadas(ver figuras A.10 e A.12), estando agora a residir dentro do co-processador. Estas estruturas de dados, são delimitadas pelas directivas do pré-processador `#ifdef __OREK_COPROCESSOR__` e `#endif // __OREK_COPROCESSOR__`.

Novamente, uma vez que o escalonamento das tarefas, a verificação das condições de preempção e a activação das tarefas aperiódicas para serviço de interrupções são completamente realizadas em *hardware* no co-processador genérico, nesta implementação existe apenas uma função invocada em caso de excepção para comutação da tarefa em execução. A função usada para este efeito possui o seguinte protótipo:
`uint COReKKern::Interruption()`
O ficheiro de interface `orek_CoP_Definitions.h` representado na figura A.21, o qual contém a interface de programação do co-processador genérico, implementa macros de simplificação no acesso às primitivas do co-processador, por forma a simplificar todas as ordens de escrita e leitura nos registos, por macros de acesso mais simples.

A.5 Utilização em Aplicações

Finalmente e para concluir a descrição do executivo *OReK*, falta apenas descrever a sua utilização. O executivo *OReK* é disponibilizado na forma de ficheiros objecto ou bibliotecas, o que significa que deve ser associado com a aplicação durante a geração do módulo executável.

A.5.1 Ficheiros a Incluir

Tal como já foi referido acima, a construção de um sistema consiste na criação de um módulo executável monolítico que integra, quer a implementação das tarefas, quer o executivo de tempo-real. Do ponto de vista do utilizador, importa considerar os seguintes ficheiros:

Ficheiros de interface:

- `OReK.h` - ficheiro geral de interface que deve ser incluído nos módulos com código fonte da aplicação onde são invocadas funções do executivo.
- `MB-PPC.h` - ficheiro geral de interface que deve ser incluído nos módulos com código fonte da aplicação onde são invocadas funções das plataformas *MB-SoC* e *PPC-SoC*.

Uma das seguintes bibliotecas, consoante a implementação desejada:

- `OReK-PC.lib` - ficheiro de implementação da versão *Intel x86/MSDOS* que deve ser associado com a aplicação no caso da plataforma de implementação *Intel x86*.
- `OReK-ARPA-Sw.a` - ficheiro de implementação da versão *ARPA-MT* sem suporte do co-processador *Cop2-OSC* e que deve ser associado com a aplicação no caso da plataforma de implementação *ARPA-Sw*.
- `OReK-ARPA-C2.a` - ficheiro de implementação da versão *ARPA-MT* com suporte do co-processador *Cop2-OSC* e que deve ser associado com a aplicação no caso da plataforma de implementação *ARPA-C2*.
- `OReK-MB.a` - ficheiro de implementação da versão *MB-SoC* que deve ser associado com a aplicação no caso da plataforma de implementação *MB-SoC*.
- `OReK-PPC.a` - ficheiro de implementação da versão *PPC-SoC* que deve ser associado com a aplicação no caso da plataforma de implementação *MB-SoC*.

A.5.2 Requisitos de Memória

Os requisitos de memória do executivo *OReK* são aproximadamente os seguintes:

- Versão *OReK-ARPA-Sw* - $20K + (48 \times \#Tarefas) + (16 \times \#Semáforos)$ bytes.
- Versão *OReK-ARPA-C2* - $10K + (16 \times \#Tarefas) + (8 \times \#Semáforos)$ bytes.
- Versão *OReK-MB* - $50K + (92 \times \#Tarefas) + (32 \times \#Semáforos)$ bytes.
- Versão *OReK-PPC* - $80K + (64 \times \#Tarefas) + (32 \times \#Semáforos)$ bytes.

Na plataforma *ARPA* a compilação do executivo *OReK* e das aplicações é suportado por um conjunto de ficheiros de projecto (*Makefiles*) desenvolvidas com o objectivo de simplificar este procedimento.

Nas plataformas *MB-SoC* e *PPC-SoC* a compilação do executivo *OReK* é gerida automaticamente por ficheiros (*Makefiles*) gerados pela ferramenta *EDK* (*Embedded Development Kit*) da *Xilinx* [Xil08b].

A.5.3 Exemplo

A figura A.22 ilustra um exemplo de um sistema de tempo-real implementado com o executivo *OReK*. A sua prototipagem foi realizada numa *FPGA XC2VP30-7ff896* da família *Virtex-II Pro* da *Xilinx* [Dig08].

O sistema possui cinco tarefas idênticas (diferentes instâncias da classe `CDemoTask`). Cada tarefa inverte e envia dados via porta série, com as características temporais especificadas pelo utilizador.

A definição da tarefa `CDemoTask` encontra-se na primeira metade da figura A.22 e possui dois atributos, um construtor e o método de entrada. O primeiro atributo é uma máscara usada na inversão do bit sobre o qual a tarefa opera. O segundo atributo é uma cópia do valor actual da saída de dados da porta série. No construtor, com base no parâmetro fornecido pelo utilizador e que identifica o bit sobre o qual a tarefa actua, é calculada a máscara referida acima. No método de entrada é lido o valor actual de todos os bits de dados da porta, invertido o bit pretendido e enviado o novo valor via porta série.

A região crítica da tarefa está protegida por um semáforo binário que garante o seu acesso em regime de exclusão mutua. Desta forma cada tarefa apenas envia invertido o bit que lhe corresponde, e por conseguinte, é assegurado o correcto funcionamento do sistema de tempo-real.

O programa principal (função `main`) começa por instanciar cinco objectos (`task1` a `task5`). Seguidamente inicializa o executivo *OReK* (função `COReKKern::Initialize(...)`). Os argumentos desta função correspondem aos seguintes parâmetros de configuração:

- Número máximo de tarefas - 10.
- Número máximo de semáforos - 2.
- Frequência do sistema - 100000 KHz.
- Resolução temporal - 100 micro-segundos.

Em caso de erro na inicialização o programa termina, caso contrário são criadas as tarefas. A existência de funções para criar as respectivas tarefas simplifica a escrita do construtor da tarefa, uma vez que os parâmetros temporais das tarefas são passados directamente ao núcleo. Além disso, facilita também a sinalização de situações de erro.

No exemplo em concreto são ilustrados os diversos tipos de tarefas suportadas pelo executivo *OReK*, estas são de tempo-real não críticas (periódicas ou aperiódicas), tempo-real críticas (periódicas ou aperiódicas) e ordinárias.

```
// Includes
#include <stdio.h>
#include "OReK.h"
#if defined (__MICROBLAZE__) || defined (__PPC__)

    #include "MB-PPC.h"

#endif // __MICROBLAZE__ || __PPC__

// Defines
#define SYS_FREQ            100000  //(Khz)
#define TICK_RESOLUTION     100     //(us)
#define MAX_TASKS           10
#define MAX_SEMAS           2

#define RS232_UART_BASEADDR 0x40600000


class CDemoTask : public COReKTask
{
public:
    CDemoTask(uchar bit)
    {
        m_xorValue = 1 << bit;
    }


public:
    virtual void Main()
    {
        // Lock sema
        DemoTaskSema.Wait();

        m_portValue ^= m_xorValue;

        XUartLite_SendByte(RS232_UART_BASEADDR, m_portValue);

        // Unlock sema
        DemoTaskSema.Signal();
    }

protected:
    uchar m_xorValue;
    static uchar m_portValue;
};

uchar CDemoTask::m_portValue = 0x00;

COReKSema DemoTaskSema;


// main
int main(int argc, char* argv[])
{

    CDemoTask task1(0);
    CDemoTask task2(1);
    CDemoTask task3(2);
    CDemoTask task4(3);
    CDemoTask task5(4);
```

```
    print("\r\nInitializing OReK ...");
    if (COReKKern::Initialize(MAX_TASKS, MAX_SEMAS,
         SYS_FREQ, TICK_RESOLUTION) != OREK_INFO_SUCCESS)
    {
        print("Failed\r\n");
        return 1;
    }
    print("0K\r\n");

    print("Creating tasks ...\r\n");
    if ((task1.CreateSoftRTPer(2, 2) != OREK_INFO_SUCCESS)    ||
        (task2.CreateSoftRTApe(16, 2) != OREK_INFO_SUCCESS)   ||
        (task3.CreateHardRTPer(4, 4, 4) != OREK_INFO_SUCCESS) ||
        (task4.CreateHardRTApe(1, 1, 1) != OREK_INFO_SUCCESS) ||
        (task5.CreateNonRT(7, true) != OREK_INFO_SUCCESS))
    {
        print("Failed\r\n");
        COReKKern::ShutDown();
        return 2;
}
    print("OK\r\n");

    print("Creating Semaphores\r\n");
    DemoTaskSema.Create();

    print("Registering Tasks\r\n");
    DemoTaskSema.RegisterTask(task1);
    DemoTaskSema.RegisterTask(task2);
    DemoTaskSema.RegisterTask(task3);
    DemoTaskSema.RegisterTask(task4);
    DemoTaskSema.RegisterTask(task5);

    print("Enabling Semaphores\r\n");
    if(DemoTaskSema.Enable() != OREK_INFO_SUCCESS)
    {
        print("Failed\r\n");
        COReKKern::ShutDown();
        return 3;
    }
    print("OK\r\n");
    print("StartAllTasks...\r\n");
    if(COReKKern::StartAllTasks() != OREK_INFO_SUCCESS)
    {
        print("Failed\r\n");
        COReKKern::ShutDown();
        return 4;
    }
    print("OK\r\n");

    while (COReKKern::GetTickCount() < 10000u)
    {
        // Idle time
    }

    print("Exiting\r\n");
    return COReKKern::ShutDown();
}
```

Figura A.22: Exemplo de um sistema implementado sobre o executivo *OReK*.

No caso das tarefas periódicas de tempo-real não críticas, são especificados os parâmetros período e fase inicial. Para as tarefas aperiódicas de tempo-real não críticas é explicitado o intervalo mínimo entre activações consecutivas (*MIT*) e o número da respectiva entrada do sinal interrupção.

Nas tarefas periódicas de tempo-real críticas são detalhados os parâmetros período, *deadline* relativa e fase inicial. Relativamente às tarefas aperiódicas de tempo-real críticas é definido o *MIT*, a *deadline* relativa e ainda a fase inicial.

Por último, para as tarefas ordinárias de baixa prioridade, os parâmetros necessários a especificar são a sua prioridade base e se estas executam uma única vez e terminam (*single shot*) ou ficam a executar durante o tempo livre do processador.

Seguidamente é criado o semáforo e são registadas todas as tarefas que acedem ao mesmo recurso partilhado do sistema. Este registo é necessário devido à utilização da política *SRP*. Esta necessita de ter conhecimento das características temporais das tarefas para calcular o respectivo tecto.

De seguida o semáforo é activado e em caso de sucesso segue-se o arranque de todas as tarefas. A partir deste momento as tarefas são executadas concorrentemente com o programa principal.

Finalmente, quando tiverem sido completados 10000 ciclos de execução *(ticks)*, o programa principal chama a função de terminação do executivo, terminando também de seguida.

Bibliografia

[AGP03] Luís Almeida, Bruno Gravato, and Bruno Pereira. ReTMiK - Real-Time Micromouse Kernel. http://sweet.ua.pt/~lda/retmik/retmik.html, 2.edition, 2003.

[Alt] Altera. Stratix ii. http://www.altera.com/products/devices/stratix-fpgas/stratix-ii/stratix-ii/st2-index.jsp.

[APA+05] David Andrews, Wesley Peck, Jason Agron, Keith Preston, Mike Finley, and Ron Sass. hthreads: A hardware/software co-designed multithreaded RTOS kernel. In *ETFA'05 - The 10th IEEE International Conference on Emerging Technologies and Factory Automation*, volume 2, pages 331– 338, Catania - Italy, September 2005. IEEE.

[ASRdO07] António Manuel de Brito Ferrari Almeida Arnaldo Silva Rodrigues de Oliveira, Valeri Skliarov. Especialização e síntese de processadores para aplicação em sistemas de tempo-real. 2007.

[Bak91] T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99, 1991.

[BJS04] K. Baskaran, W. Jigang, and T. Srikanthan. A hardware operating system based approach for run-time reconfigurable platform of embedded devices. In *The 6th Real Time Linux Workshop*, Singapore, November 2004.

[CES71] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, 1971.

[CRL06] Sathish Chandra, Francesco Regazzoni, and Marcello Lajolo. Hardware/software partitioning of operating systems: a behavioral synthesis approach. In *GLSVLSI'06 - The 16th ACM Great Lakes symposium on VLSI*, pages 324–329, Philadelphia - PA - USA, April 2006. ACM Press.

[CYC+05] Youngchul Cho, Sungjoo Yoo, Kiyoung Choi, N.-E. Zergainoh, and Ahmed Amine Jerraya. Scheduler implementation in MP SoC design. In *ASP-DAC'05 - The Asia and South Pacific Design Automation Conference*, pages 151–156, Shanghai - China, January 2005. IEEE.

[Dig08] Digilent, Inc. Digital Design Engeneer's Source, April 2008. www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,794&Prod=XUPV2P.

[dS08] Nelson José Valente da Silva. Executivo de tempo-real para processadores embutidos em fpga. Master's thesis, 2008.

[DWX+05] Qingxu Deng, Shuisheng Wei, Hai Xu, Yu Han, and Ge Yu. A reconfigurable rtos with hw/sw co-scheduling for sopc. pages 6 pp.–, Dec. 2005.

[IW04] Spencer Isaacson and Doran Wilde. The task-resource matrix: Control for a distributed reconfigurable multi-processor hardware RTOS. In *ERSA'04 - The International Conference on Engineering of Reconfigurable Systems and Algorithms*, pages 130–136, Porto - Portugal, August 2004. CSREA Press.

[LA08] Paulo Pedreiras Luis Almeida. Sistemas de tempo-real : Slides aula 7. IEETA/Universidade de Aveiro - Portugal, 2008. http://www.ieeta.pt/lse/str/str-0708/slides-aula7.pdf.

[Met05] Alexander Metzner. Predictable and efficient architectures for real-time system synthesis. In *RTSS'05 - The 26th IEEE Real-Time Systems Symposium - Work in Progress session*, Miami - FL -USA, December 2005. IEEE Computer Society.

[MGBN06] A. Mitra, Zhi Guo, A. Banerjee, and W. Najjar. Dynamic co-processor architecture for software acceleration on csocs. pages 127–133, Oct. 2006.

[MI05] Vincent John Mooney III. Hardware/software partitioning of operating systems. In *IEE Proceedings on Computers and Digital Techniques*, volume 152, pages 167–182. IEE, March 2005.

[Mic] Micrium. Microc/os2. http://micrium.com/page/products/rtos/os-ii.

[MNM+04] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. Bartic, W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Run-time support for heterogeneous multitasking on reconfigurable SoCs. *Integration, the VLSI Journal*, 38(1):107–130, October 2004.

[NA07] S. Nordstrom and L. Asplund. Configurable hardware/software support for single processor real-time kernels. pages 1–4, Nov. 2007.

[NL05] Susanna Nordström and Lennart Lindh. Application specific real-time microkernel in hardware. In *RTC'05 - The 14th IEEE-NPSS Real Time Conference*, Stockholm - Sweden, June 2005. IEEE.

[Rea05] RealFast. *Sierra - Real-Time Kernel in Hardware - Operating System Accelerator*. RealFast, http://www.realfast.se, 1.0 edition, January 2005.

[SHC07] Moonvin Song, Sang Hoon Hong, and Yunmo Chung. Reducing the overhead of real-time operating system through reconfigurable hardware. *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 311–316, Aug. 2007.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[SS05] P. Samson and P. Sinha. Hardware acceleration of deadlock avoidance and detection in real-time operating systems. pages 429–433, July 2005.

[SVCG99] S. Saez, J. Vila, A. Crespo, and A. Garcia. A hardware scheduler for complex real-time systems. volume 1, pages 43–48 vol.1, Aug. 1999.

[UCR] UCR. Riverside Optimizing Compiler for Configurable Computing. http://www.cs.ucr.edu/ roccc/ROCCC/Home.html.

[WA06] Kack Whitham and Neil Audsley. MCGREP - a predictable architecture for embedded real-time systems. In *RTSS'06 - The 27th IEEE Real-Time Systems Symposium*, pages 13–24, Rio de Janeiro - Brazil, December 2006. IEEE Computer Society.

[Xil07] Xilinx. Processor local bus v4.6, 2007. http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.

[Xil08a] Xilinx, Inc. MicroBlaze Processor, March 2008. Xilinx UG081 (v9.0) MicroBlaze Processor, Reference Guide.

[Xil08b] Xilinx, Inc. Platform Studio and the EDK, March 2008. http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm.

[Xil08c] Xilinx, Inc. PowerPC 405 Processor, April 2008. Xilinx UG011 PowerPC Processor, Reference Guide.

[YW04] Matthew Young and Doran Wilde. RTPOS: A customizable hardware/software real time operating system on a programmable chip. http://www.ee.byu.edu/faculty/wilde/RTSOPC/rtsopc.html, August 2004.